PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UNA INTERFAZ JAVA EE PARA UNA APLICACIÓN MULTIHITOS

Autor: Alfredo Gordo García

Tutor: Pablo Basanta Val

Leganés, Junio de 2013

Título: Diseño e implementación de una interfaz Java EE para una aplicación multihitos
Autor: Alfredo Gordo García
Director: Pablo Basanta Val

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

SECRETARIO VOCAL PRESIDENTE

Agradecimientos

En primer lugar, quisiera dar las gracias a mi tutor Pablo, por los consejos y directrices que me han permitido llevar este proyecto a buen puerto. Del mismo modo, agradecer a todos aquellos profesores gracias a los cuales he podido adquirir los conocimientos y experiencia necesaria para afrontarlo.

También quisiera agradecer sobre todo a mi familia, en especial a mis padres y a mi hermano, por darme todo lo necesario para poder completar mis estudios universitarios y porque siempre han creído en mí. Su ayuda y apoyo ha sido fundamental durante todos estos años.

Además quiero dar las gracias a mis amigos y compañeros de la universidad y la residencia, por todos esos buenos ratos que han ayudado a amenizar los momentos más difíciles. A Abel, por todas esas incontables horas que nos pasamos con las prácticas.

Gracias también a mis amigos del pueblo y de mi tierra, por esos fantásticos veranos que ayudaban a desconectar por un tiempo de los duros estudios.

.

Resumen

Este proyecto de fin de carrera surge dentro del entorno de un proyecto de innovación docente, consistente en el desarrollo de un mecanismo de *hitos* mediante la participación activa de alumnos y profesores que pretende mejorar la eficacia del aprendizaje.

Hasta el momento se había probado esta estrategia en algunas clases, sin más herramientas que papel y bolígrafo. La idea que da origen a este PFC consiste en diseñar e implementar una interfaz Web para este sistema, proporcionando al mismo un acceso electrónico más atractivo, cómodo y seguro que mediante el método tradicional.

La tecnología escogida para el desarrollo de la aplicación Web es la parte empresarial de java, *Java EE***.** En la parte del servidor, se ha usado *Apache Tomcat 7.0* como contenedor de *servlets* y *JSP*, y para el almacenamiento de datos persistente una base de datos *MySQL*. El cliente será, sencillamente, un navegador Web convencional. Además, para facilitar la portabilidad y flexibilidad se ha decidido desarrollar el sistema dentro de una máquina virtual mediante el software *VirtualBox*. Todos los componentes utilizados están dentro del marco de la filosofía de libre distribución (*open source framework)* y son fácilmente descargables e instalables.

La memoria comienza con una breve introducción y motivación del problema que se pretende resolver, seguida de unos capítulos introductorios a las tecnologías utilizadas para la resolución del mismo. Posteriormente se centra en el diseño y desarrollo de la aplicación en sí, resultados de pruebas de funcionamiento y rendimiento, y finalmente se incluyen unas conclusiones y un presupuesto.

Palabras clave: hitos, aprendizaje, innovación docente, desarrollo Web, JavaEE, servlets, JSPs, MySQL, Apache Tomcat, VirtualBox.

Abstract

This master thesis arises inside the environment of a teaching innovation project which involves the development of a *milestones* mechanism through active participation of students and teachers that aims to improve the efficiency of the learning.

Until now, this strategy had been proved in some classes, no more tools than paper and pen. The idea that gives rise to this master thesis is to design and implement a Web interface for this system, providing a electronic access more atractive, comfortable and secure than using the conventional method.

The chosen technology for the Web application development is the enterprise part of Java, *Java EE*. On the server side, *Apache Tomcat Servlet 7.0* has been used as a *servlets* and *JSPs* container, and for the persistent data storage, a *MySQL* database. The client is simply a conventional Web browser. Moreover, in order to facilitate portability and flexibility, it has been decided to develop the system within a virtual machine using VirtualBox software. All components used are within the framework of a free distribution philosophy and are easily downloadable and installable.

The report begins with a brief introduction and motivation of the problem want to resolve, followed by some introductory chapters to the technologies used to solve it. Later, it focus on the design and development of the application itself, operational results and performance tests. Finally, it will be included conclusions and the project budget.

Keywords: milestones, learning, teaching innovation, Web development, JavaEE, servlets, JSPs, MySQL, Apache Tomcat, VirtualBox.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

La docencia universitaria tradicional está sufriendo modificaciones debido a la adaptación de la enseñanza a los principios del Espacio Europeo de Educación Superior (EEES, [1]). Con el Tratado de Bolonia ([2]), la concepción del profesor como figura única y fundamental en la transmisión del conocimiento ante unos estudiantes con un papel pasivo se va difuminando. El profesor se convierte más bien en un tutor/supervisor y se pretende dar mucho más protagonismo al estudiante.

En este marco, los proyectos de innovación docente ([3]) adquieren cada vez más importancia en el desarrollo de metodologías innovadoras que permitan aprendizajes más eficaces y atractivos con la participación activa del estudiante en la construcción del conocimiento. El proyecto de innovación docente que a su vez da origen a este Proyecto de Fin de Carrera se desarrolla en la Universidad Carlos III con el título "*Mejora de la Eficacia del Aprendizaje mediante un Mecanismo de Hitos Participado por Alumnos y Profesores*".

Dentro de este proyecto, los hitos se usan como una herramienta general que puede servir para lograr un aprendizaje más efectivo. Cada día, el alumno puede estimar su avance dentro de las distintas tareas y compararlo con el de sus compañeros. De esta manera se motiva y se hace más partícipe al alumno, teniendo que autoevaluar el cumplimiento de los hitos, haciéndolo más consciente y autónomo en su aprendizaje. Para los profesores, son una fuente de información importante para la mejora continua de sus asignaturas, pues le permiten producir clases más adaptadas a las necesidades concretas de cada grupo. También puede ayudar a realizar un mejor uso de las tutorías, convocando a aquellos alumnos que se han quedado más rezagados.

Esta idea se ha empezado a desplegar en algunos cursos sin más herramientas que papel y bolígrafo. El profesor creaba sus encuestas en forma de tablas que los alumnos iban completando. Los usos pueden ser diversos: control de asistencia y evolución en clase, control del avance en las prácticas de laboratorio, o control del trabajo en equipo dentro de un proyecto, entre otros.

La experiencia en general ha sido satisfactoria y esto es lo que ha dado lugar a la idea de la que nace este PFC: diseñar e implementar una interfaz Web para este mecanismo. Se pretende desarrollar una herramienta que proporcione un acceso electrónico más atractivo, cómodo y seguro ([4]), al mismo tiempo que no introduzca gran sobrecarga en los alumnos ni en los profesores.

La aplicación Web se desarrollará con la tecnología *Java Enterprise Edition (JEE* [5]*)*, permitiendo un acceso rápido a los alumnos para completar las encuestas, y un acceso más elaborado para los profesores que podrán crear y configurar sus encuestas según sus necesidades. La veracidad de las firmas estará garantizada mediante la autenticación contra la base de datos de la universidad a través de LDAP ([6],[7]). De todas formas, la configuración de acceso con LDAP podrá ser modificada por el administrador para poder adaptar el sistema a otros entornos distintos de la UC3M y aportar mayor flexibilidad al sistema.

Igualmente, por flexibilidad y comodidad, se ha decidido realizar la implementación de la aplicación dentro de una máquina virtual con sistema Linux, mediante el software *VirtualBox* ([8]). Como servidor Web y contenedor de servlets y JSPs se ha escogido *Apache Tomcat 7.0* ([9]), y para el almacenamiento de la información una base de datos *MySQL* ([10]). Como cliente Web basta con un navegador normal. Todos estos componentes son de código abierto y fácilmente descargables, instalables y configurables. Serán explicados posteriormente con más profundidad. Para hacerse una primera idea global del sistema desarrollado, se ilustra en la Figura 1 un esquema general.

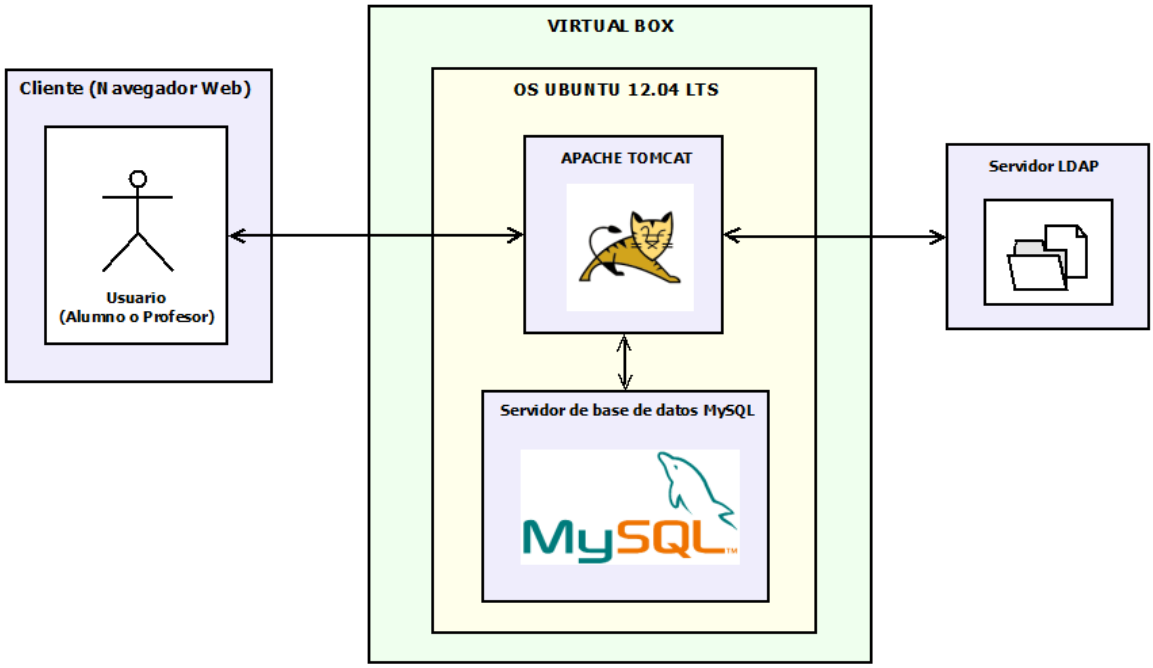


Figura 1. Esquema general de la aplicación desarrollada

En las siguientes secciones de este capítulo introductorio, se explican los objetivos marcados para este PFC, las distintas fases en las que se ha dividido su desarrollo, y los medios o recursos necesarios para su realización. Para finalizar, se incluye un esquema de la estructura de la memoria.

1.2 Objetivos

El objetivo principal de este PFC es la realización de una interfaz Web para el mecanismo de hitos mencionado, mediante el uso de la tecnología Java EE. Los distintos subobjetivos marcados con el fin de lograr el desarrollo de la aplicación son:

1. **Despliegue de una máquina virtual donde implementar el sistema**

   Puesto que el tipo de aplicación a desarrollar no va a exigir un consumo de recursos elevado ni requiere un rendimiento excepcional, una manera cómoda de desarrollar el proyecto es implementarlo dentro de una máquina virtual ([11], [12]). El software VirtualBox de Oracle está disponible para la mayoría de sistemas operativos y permite implantar una máquina virtual de manera sencilla. Esto aporta portabilidad, pues permite desarrollar el proyecto en cualquier ordenador con sólo copiar el archivo que representa el disco virtual, y también aporta flexibilidad, pues se puede configurar en qué tipo de máquina (características hardware y OS) queremos desarrollar nuestro proyecto. Se ha decidido desarrollar el proyecto en un sistema Linux ([13]), ya que es compatible y se integra muy bien con todos los componentes necesarios para el desarrollo de nuestra aplicación. La distribución y versión escogida es Ubuntu 12.04 LTS ([14]), por tratarse de la última versión estable con una amplia comunidad de soporte.

2. **Estudio de Java EE**

   Se trata de analizar el estado de desarrollo de esta tecnología, viendo las diferentes posibilidades que ofrece la plataforma Java EE para el desarrollo del sistema, esto es, qué componentes se adaptan mejor a las necesidades de la aplicación que se quiere implementar.

3. **Análisis e instalación de la herramienta Apache Tomcat**

   Para poder ejecutar soluciones Web basadas en Java EE, se necesita un servidor Web ([15]) con capacidad de tratar componentes de la especificación Java EE. Con este fin, se utilizará la herramienta de código libre Apache Tomcat. Esta herramienta reúne las funcionalidades de un servidor Web como son las que se refieren a la gestión del protocolo HTTP ([16],[17]), con las de lo que se denomina un contenedor Web (*Web Container)*, encargado de gestionar los componentes Java EE. Se utilizará la última versión disponible, Apache Tomcat 7.0.

4. **Análisis e instalación del Sistema Gestor de Base de Datos MySQL**

   La mayoría de las aplicaciones precisa de un nivel de datos para almacenar la información de manera persistente y coherente. Una base de datos relacional ([18]) se adapta muy bien a las necesidades concretas de la aplicación que se pretende realizar. Se ha decidido utilizar el Sistema Gestor de Base de Datos ([19]) MySQL Community Server por tratarse de una herramienta de código abierto muy popular y fácilmente configurable.

5. **Diseño e implementación de la aplicación**

   Una vez cumplidos los objetivos anteriores estaremos en disposición de diseñar e implementar la aplicación. Se editarán las clases java y otros ficheros necesarios y se desplegarán dentro de la estructura de Apache Tomcat.

6. **Pruebas de funcionalidad y rendimiento**

   El último objetivo consistirá en comprobar el funcionamiento de la aplicación para depurar errores y hacer pruebas de rendimiento para medir tiempos de respuesta.

1.3 Fases del desarrollo

La duración de la realización del proyecto ha sido de unos seis meses. A continuación se resumen las fases en las que se ha dividido el desarrollo del mismo para llevarlo a cabo, y la duración aproximada de cada una de ellas.

1. **Planteamiento de la necesidad y de cómo resolverla (5 días)**

   Reunido con el equipo de innovación docente, se planteó la utilidad de desarrollar la aplicación dentro del marco del proyecto *multihitos*. Se acordaron las especificaciones y se decidieron las tecnologías concretas con las que implementar la solución. Se estableció un plan de trabajo para el PFC.

2. **Estudio de JavaEE y de las distintas herramientas a utilizar (1 mes)**

   Se realizó un análisis de la tecnología JavaEE, que es la principal en el desarrollo de la aplicación, centrándose en los componentes de la misma que finalmente fueron necesarios: *servlets* y *JSPs.* De igual modo se realizó un estudio del software concreto que se necesita : VirtualBox, Apache Tomcat, y MySQL Server.

3. **Diseño del sistema (15 días)**

   Se diseñó el esquema relacional de la base de datos necesario para la capa de datos de nuestra aplicación, y un pequeño esquema del sitio Web. Se diseñaron el

diagrama de casos de uso de la herramienta, diagrama de clases, diagramas de interacción y diagrama de despliegue ([20]).

4. **Configuración del entorno de trabajo (5 días)**

Se descargaron, instalaron y configuraron las herramientas necesarias: la máquina virtual con OS Ubuntu, y dentro de la misma el SDK ([21]) de Java, el servidor Apache Tomcat y el SGBD MySQL Server. Se implementó el esquema relacional de datos diseñado, dejándolo preparado para interacciones con la aplicación.

5. **Implementación (2 meses)**

En esta fase, se escribió todo el código necesario para el funcionamiento de la aplicación y se desplegó dentro de la estructura de directorios de Apache Tomcat: servlets, JSPs, clases java, ficheros XML ([22]) y hojas de estilo ([23]).

.

6. **Refinamiento de la aplicación y pruebas (15 días)**

Se probaron las distintas funcionalidades de la aplicación con varios ejemplos y se corrigieron errores. Se hicieron pequeñas modificaciones y añadidos. Se realizaron pruebas del tiempo de respuesta de las distintas operaciones.

7. **Redacción de la memoria (1 mes y medio)**

Es necesaria la redacción de una memoria formal donde se documenta todo el trabajo realizado y las tecnologías empleadas.

8. **Preparación de la presentación (7 días)**

Se ha diseñado y ensayado una breve presentación con diapositivas sobre el PFC con vistas a exponerlo ante el tribunal, pretendiendo que sea clara, amena y concisa.

1.4 Medios empleados

Debido a la propia naturaleza del proyecto, no son necesarios muchos ni costosos recursos. Se resumen a continuación:

Hardware

- PC con conexión a red: Cualquier ordenador portátil o de sobremesa es suficiente para los requerimientos de este proyecto. Con un sólo ordenador ha sido suficiente, teniendo en local servidor, base de datos y cliente, todo dentro de la misma máquina virtual. Este escenario es válido en un entorno de pruebas. Aunque en un entorno más real el servidor, la base de datos y los clientes estarían muy probablemente en máquinas distintas, el sistema puede adaptarse sencillamente cambiando direcciones IP ([24]) y números de puerto ([25]). La conexión a Internet se hace necesaria para

descargar el software, buscar información y para la parte de autenticación contra la base de datos de la universidad a través de LDAP.

Software

El software necesario ya se ha mencionado en las secciones anteriores. Todos los componentes utilizados son gratuitos. Los recordamos:

- SO Linux: Ubuntu versión 12.04 LTS ([14]). En realidad, el sistema operativo de nuestro PC no es excesivamente relevante siempre que soporte VirtualBox, ya que se implementa el sistema en una máquina virtual.

- VirtualBox versión 4.1.12 ([8]): necesario para lanzar nuestra máquina virtual.

- Java EE 6 SDK: el Software Development Kit de Java es necesario para trabajar con servlets, JSPs y clases java convencionales. Incluye todas las herramientas de desarrollador necesarias (como el compilador y el *debugger*) y el *Java Runtime Enviroment* (JRE) para ejecutar nuestras aplicaciones Java. Se puede descargar de [5].

- Apache Tomcat 7.0 ([9]): servidor Web y contenedor de componentes JEE.

- MySQL Comunnity Server 5.5 ([10]): para el nivel de datos de la aplicación.

1.5 Estructura de la memoria

Para facilitar la lectura de la memoria, se incluye a continuación un esquema de la estructura de la misma:

- Bloque I: Introducción
    - Capítulo 1. Introducción y objetivos

      Capítulo inicial donde se presenta la motivación del proyecto y se establecen los objetivos a alcanzar, dando una idea general del trabajo que se va a realizar. Se describen las fases de desarrollo y los recursos software y hardware necesarios, así como un esquema de la estructura de esta memoria.

- Bloque II: Estado del arte
    - Capítulo 2. El sistema multihitos

      En este capítulo se explica en profundidad en qué consiste la estrategia de los *hitos*, y cómo se está llevando a cabo hasta el momento en el entorno de la universidad.

- Capítulo 3. La plataforma Java EE

  Estudio y análisis de la utilidad y estado de esta tecnología, centrándonos en los componentes que son más necesarios para el proyecto: *servlets* y *JSPs.*

- Capítulo 4. El servidor Apache Tomcat

  En este capítulo se estudia el servidor Web Apache Tomcat, su funcionalidad y características, y se explicará cómo configurar una aplicación dentro de su estructura de directorios.

- Bloque III: Diseño e implementación de la aplicación y pruebas.
  - Capítulo 5. Diseño de la aplicación

    Se detalla el diseño de las distintas partes de la aplicación: nivel de datos e interfaz Web. Diagramas tipo UML ([20]) que explican el funcionamiento de la aplicación.

  - Capítulo 6. Implementación

    En este capítulo se describen los distintos módulos software desarrollados y la interacción entre los mismos para lograr la funcionalidad deseada.

  - Capítulo 7. Pruebas

    Aquí se describen y presentan los resultados de las distintas pruebas realizadas para verificar funcionamiento y tiempos de respuesta de la aplicación.

- Bloque IV: Conclusiones y líneas futuras de trabajo
  - Capítulo 8. Conclusiones y líneas futuras de trabajo

    Para finalizar, se muestran unas conclusiones sobre el trabajo realizado y el grado de consecución de los objetivos propuestos, así como las posibles líneas futuras de trabajo en esta temática.

- Bloque V: Apéndices
  - Apéndice A: Presupuesto
  - Apéndice B: Instalación y configuración de las herramientas utilizadas
  - Apéndice C: Aspectos específicos de implementación
  - Apéndice D: Tablas de prueba
  - Apéndice E: Referencias

Capítulo 2

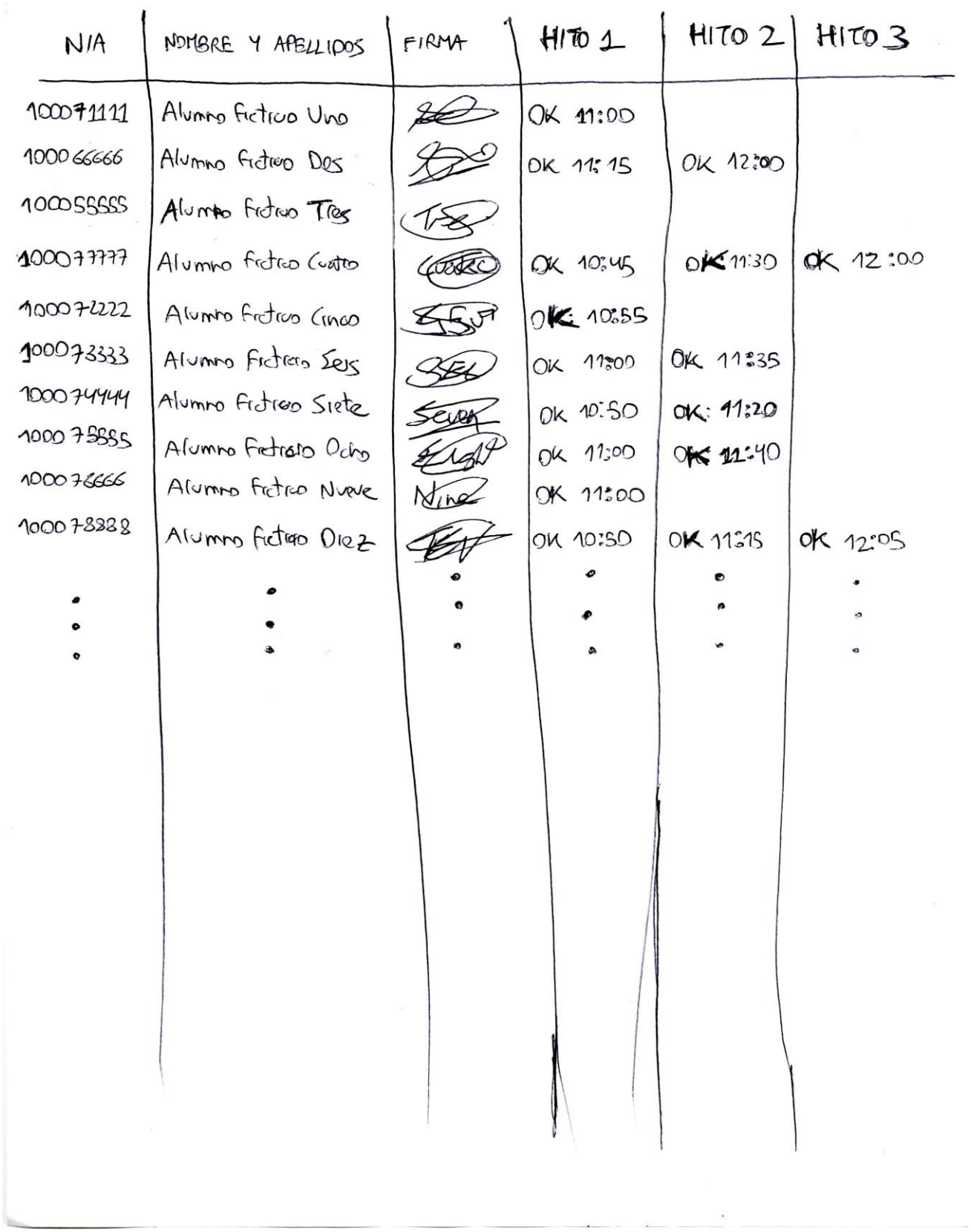
El sistema multihitos

2.1 Introducción

La estrategia de *hitos* surge como consecuencia de la nueva metodología de clases que supone la adaptación al EEES y al Tratado de Bolonia. Las técnicas tradicionales han quedado anticuadas, y resulta interesante desarrollar nuevas metodologías más activas y dinámicas en la enseñanza universitaria, que involucren al alumno y le hagan sentirse partícipe de su propio aprendizaje. De esta manera se intenta aprovechar mejor los recursos y obtener mejores resultados. En este contexto, nace la idea de realizar un control de las diversas actividades que se llevan a cabo en los cursos, un control en el que participen tanto profesores como alumnos, y aporte cierta información valiosa para ambas partes. El profesor puede establecer ciertos puntos de referencia u objetivos (*hitos*) dentro de las actividades, y el propio alumno será el encargado de determinar o marcar cuando va alcanzando los hitos en cada actividad. Esta idea se plasma de modo formal en el proyecto de innovación docente de título *"Mejora de la Eficacia del Aprendizaje mediante un Mecanismo de Hitos Participado por Alumnos y Profesores"* desarrollado en la Universidad Carlos III de Madrid, que da origen a este PFC.

2.2 ¿Qué es un hito?

El concepto de *hito* puede parecer un poco abstracto en teoría, pero se entiende enseguida al llevarlo a la práctica. Por ejemplo, pensemos en una tarea de una asignatura, como un trabajo o una práctica de laboratorio, que se compone de tres subtareas bien diferenciadas. En este caso los hitos podrían ser, directamente, la realización de manera satisfactoria de cada una de las tres subtareas. Para llevar el cabo el control de dicha tarea, lo más sencillo es la realización de una encuesta en papel por parte del profesor, que los alumnos deberán ir completando. Y la forma más visual de llevar a cabo esta encuesta es mediante una tabla. El aspecto de una encuesta generada de este modo puede verse en la Figura 2.



NIA	NOMBRE Y APELLIDOS	FIRMA	HITO 1	HITO 2	HITO 3
100071111	Alumno Ficticio Uno		OK 11:00		
100066666	Alumno Ficticio Dos		OK 11:15	OK 12:00	
100055555	Alumno Ficticio Tres				
100077777	Alumno Ficticio Cuatro		OK 10:45	OK 11:30	OK 12:00
100072222	Alumno Ficticio Cinco		OK 10:55		
100073333	Alumno Ficticio Seis		OK 11:00	OK 11:35	
100074444	Alumno Ficticio Siete		OK 10:50	OK 11:20	
100075555	Alumno Ficticio Ocho		OK 11:00	OK 11:40	
100076666	Alumno Ficticio Nueve		OK 11:00		
100078888	Alumno Ficticio Diez		OK 10:50	OK 11:15	OK 12:05
...	...	...	...	...	...

Figura 2. Ejemplo de hoja de hitos

2.3 Ventajas de la estrategia de hitos

En la hoja de hitos anterior, los campos NIA y firma sirven para verificar la identidad de los estudiantes. Además, los alumnos indican su nombre, y van marcando los hitos en el momento de su consecución, indicando también la hora. De esta manera, los alumnos pueden ir midiendo su avance dentro de la tarea, y comparar su progreso con el del resto de sus compañeros. Este ejercicio de autoevaluación del cumplimiento de hitos es una manera de motivar y fomentar la participación del alumno, haciéndolo más consciente de su aprendizaje y aumentando su autonomía. Además, el procedimiento para rellenar este tipo de encuestas es muy ágil, con lo que no se incurre en una gran sobrecarga ni pérdida de tiempo para el alumno.

En cuanto al profesor, las encuestas de hitos le proporcionan una fuente de información útil tanto en la clase como posteriormente. Si, por ejemplo, los alumnos están realizando una práctica de laboratorio, el profesor puede consultar la hoja de vez en cuando para comprobar en directo el avance de sus alumnos, y ayudar a aquellos que se vayan quedando atrás. Al final de la clase, se queda con la hoja y puede hacer un análisis más detenido de la misma. Con la información de varias hojas de un mismo grupo, puede incluso estimar ciertas estadísticas o ver patrones que le pueden servir para conocer mejor la evolución de cada alumno en la asignatura de manera individual y ayudar a los más desaventajados. Si el profesor imparte la misma asignatura a varios grupos distintos, puede hacer comparaciones entre los resultados de los mismos en las encuestas, lo que le permite impartir unas clases más acordes a las necesidades concretas de cada grupo. Las encuestas también pueden servir al profesor para planificar un uso más eficiente de las tutorías, convocando a los alumnos que más lo necesiten.

2.4 Estado actual del proyecto de innovación

El mecanismo de hitos se ha empezado a utilizar en algunos cursos de la Universidad. Aunque aún no se ha desplegado totalmente en las asignaturas propuestas, se está probando su funcionamiento con fines diversos:

- Control de asistencia y evolución en clase
- Control del avance en las prácticas de laboratorio
- Control del trabajo en equipo dentro de un proyecto
- Uso eficiente de tutorías
- Construcción de conocimiento guiado mediante hitos

También se plantea la utilización del sistema de hitos como parte de la evaluación. El mecanismo sería fácilmente integrable como medio para calificar, pero todavía no se ha usado con este fin, debido a que esto exige un fuerte control de supervisión del cumplimiento de los hitos, que puede ir en detrimento de la agilidad y el dinamismo de la idea inicial.

De cara a reducir el esfuerzo de implantación, en el proyecto de innovación docente se propone, si es necesario, la generación de material sobre cómo cubrir los hitos (para los alumnos), y sobre cómo organizar los hitos para una determinada unidad didáctica o asignatura (para los profesores).

La experiencia con el proyecto está siendo satisfactoria en líneas generales, por lo que proporcionar el acceso Web sugerido en este PFC puede ser un empuje en la dirección adecuada para refinar y afianzar esta metodología en las clases. Se pretende de esta manera que el uso de la herramienta sea más atractivo y ágil tanto para alumnos como para profesores. El acceso electrónico aporta además ventajas adicionales como un mejor control de autenticación, una configuración de las encuestas más completa, y un almacenamiento de la información (encuestas) más cómodo y fiable.

Capítulo 3

La plataforma Java EE

3.1 Introducción

La plataforma Java Enterprise Edition o simplemente Java EE ([5]) es una plataforma de programación propiedad de Oracle Corporation ([26]). Informalmente conocido como Java Empresarial, permite el desarrollo y ejecución de aplicaciones diseñadas en arquitecturas de *n* capas ([27]) en un entorno distribuido. Las aplicaciones estarán desarrolladas esencialmente en el lenguaje de programación Java ([28]), apoyándose en componentes de software modulares que corren sobre un servidor de aplicaciones.

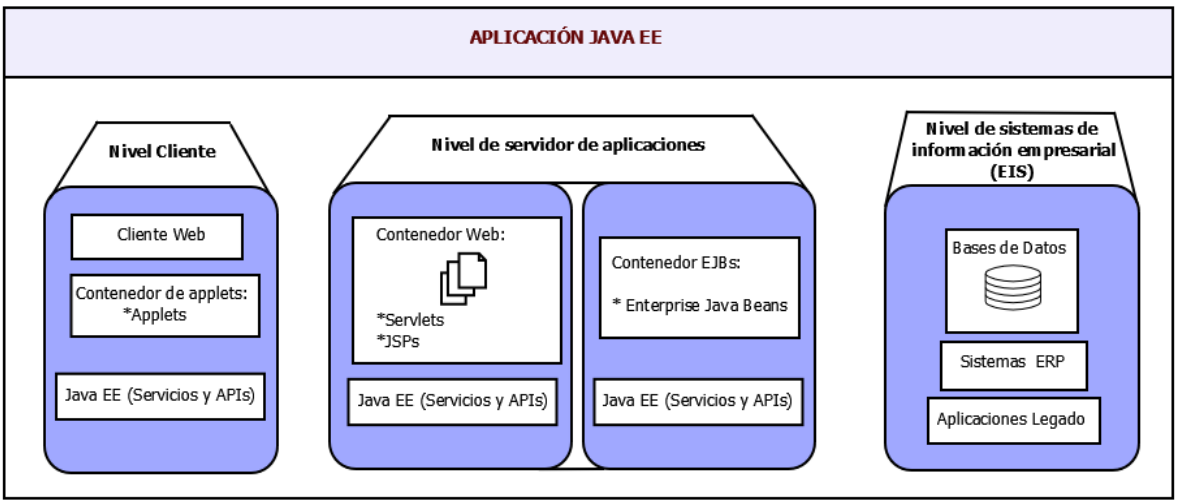


Figura 3. Arquitectura de una aplicación genérica Java EE

Como se ve en la Figura 3, los componentes se despliegan dentro de lo que se denominan *contenedores*. Los contenedores son elementos que proporcionan una interfaz entre un componente y la funcionalidad específica de más bajo nivel que soporta ese componente. Antes de que cualquier componente se pueda ejecutar, debe ser ensamblado dentro de un módulo Java EE y desplegado dentro de su contenedor. Los contenedores se encargan de tareas tales como las siguientes:

- Ofrecer un entorno de ejecución para los componentes de la aplicación.
- Gestión de los recursos y del ciclo de vida de los componentes.
- Proporcionar una vista uniforme de los servicios que requieren los componentes.
- Proporcionar herramientas de despliegue para la instalación y configuración de los componentes.

Java EE es también considerada como un estándar, ya que se puede ver como una colección de especificaciones y directivas de programación que los productos de los proveedores deben cumplir para poder declarar que sus productos son conformes a JavaEE. La especificación está dentro de *The Java Community Process* (JCP, [30]).

3.1.1 Historia de las versiones

La plataforma era conocida como como *Java 2 Platform, Enterprise Edition* or J2EE hasta la versión 1.4, a partir de la cual se denominó sencillamente *Java Platform, Enterprise Edition* o Java EE. A lo largo del tiempo, la plataforma fue extendiendo la funcionalidad del modelo estándar *Java Platform, Standard Edition* (JSE, [31]) con el desarrollo de diferentes versiones:

- J2EE 1.2 (12 de Diciembre de 1999)
- J2EE 1.3 (24 de Septiembre de 2001)
- J2EE 1.4 (11 de Noviembre de 2003)
- Java EE 5 (11 de Mayo de 2006)
- Java EE 6 (10 de Diciembre de 2009)
- Java EE 7 (12 de Junio de 2013)

Cada versión extiende la anterior con nuevos API ([32]) que aportan nuevos servicios y/o funcionalidades. Puede consultar los detalles concretos de cada versión en [5]. La versión 6 es la utilizada en el desarrollo del proyecto. Como se puede observar, Java EE es una tecnología en constante evolución, y la nueva versión Java EE 7, acaba de ser presentada en el mes de Junio de este año.

3.2 APIs y Servicios

Java EE se construye sobre la base de Java SE, añadiendo la funcionalidad y proporcionando al desarrollador los servicios necesarios para crear una aplicación de empresa con buenas características de escalabilidad, portabilidad e integración con tecnologías anteriores. Además, el servidor de aplicaciones permite manejar transacciones, seguridad, concurrencia y gestión de los componentes empleados, y de esta manera el desarrollador puede ocuparse más en la lógica de negocio de los componentes que en tareas de mantenimiento de más bajo nivel.

A continuación se repasan los servicios más usuales que se proporcionan y las APIs asociadas a dichos servicios. El listado completo de todas las APIs y su especificación en detalle puede consultarse en [33].

- **EJBs (javax.ejb.*):** en esta API se definen las clases e interfaces necesarias para manejar *los Enterprise JavaBeans* (EJBs). Se especifican las interacciones entre los EJBs y sus clientes, y entre los EJBs y el contenedor. Los EJBs proporcionan servicios de comunicación (procedimientos de invocación remota, RMI [34]), concurrencia, transacciones y control de acceso.

- **JSF (javax.faces.*):** *Java Server Faces* (JSF, [35]) proporciona un entorno para simplificar el desarrollo de interfaces de usuario en aplicaciones Java EE.

- **JMS (javax.jms):** el API de *Java Messaging Service* (JMS) proporciona una manera común para que los programas Java creen, envíen, reciban y lean los mensajes de un sistema de comunicación empresarial.

- **JWS (javax.jws.*):** el *Java Web Service* (JWS) proporciona las interfaces y elementos necesarios para que las aplicaciones interactúen con Servicios Web ([36]).

- **JPA (javax.persistence.*):** este paquete proporciona las clases e interfaces que modelan la interacción entre proveedores de persistencia, clases administradas y clientes del *Java Persistence API (JPA).* El objetivo de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional, [37]).

- **JCA (javax.resource.*):** este paquete define la *Java EE Connector Architecture* (JCA). Esta tecnología permite conectar las aplicaciones en los servidores con los sistemas de información internos de las empresas.

- **Security Service (javax.security.*):** aquí se abordan temas de seguridad como el control de acceso a recursos protegidos, autenticación y autorización, y otros.

- **Servlets y JSPs (javax.servlet.*):** esta especificación define un conjunto de APIs usadas principalmente para tratar peticiones HTTP. En el paquete se incluye también la especificación JSP para aspectos más de presentación.

- **JTA (java.transaction.*):** estos paquetes definen la *Java Transaction API* (JTA). JTA establece una serie de interfaces Java entre el manejador de transacciones y las partes involucradas en el sistema de transacciones distribuidas ([38]): el servidor de aplicaciones, el manejador de recursos y las aplicaciones transaccionales.

- **JAXP (javax.xml.*):** estos paquetes definen el API de *Java API for XML Processing* (JAXP). Es un API que se encarga de la manipulación y tratamiento de archivos XML.

Otros paquetes muy usados aunque no son exclusivos de Java EE 6 sino que pertenecen a Java SE 6 son:

- **JNDI (javax.naming.*):** definen la API de *Java Naming and Directory Interface* (JNDI). Permite a los clientes descubrir y buscar componentes y recursos a través de un nombre lógico, independientemente de la aplicación subyacente.

- **JDBC (java.sql y javax.sql):** Los paquetes java.sql y javax.sql definen el API de *Java DataBase Connectivity* (JDBC). Es un API que maneja la conectividad de los programas Java con las bases de datos, tratando de independizar las aplicaciones del SGBD concreto.

3.3 Componentes

La tecnología Java EE es una tecnología basada en componentes (módulos software) que interactúan entre sí permitiendo desarrollar aplicaciones empresariales escalables, fiables y seguras. En esta sección se estudian los componentes fundamentales de esta tecnología: *EJBs, servlets y JSPs*. Nos centraremos especialmente en servlets y JSPs, ya que son los componentes que finalmente se emplearán en la implementación de nuestra aplicación, puesto que son los más adecuados para el desarrollo de la interfaz Web (nivel de presentación).

3.3.1 Enterprise JavaBeans (EJBs)

Los EJBs ([39]) proporcionan un modelo de componentes distribuido estándar del lado del servidor. El hecho de estar basado en componentes permite que éstos sean flexibles y reutilizables. Los EJBs son el corazón de la especificación Java EE, y desde el inicio han sufrido una evolución constante pasando por varias versiones. La versión actual es EJB 3.1 (Diciembre de 2009), perteneciente a Java EE 6. Los objetivos principales de esta arquitectura de componentes son:

- Proporcionar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia y seguridad) para facilitar el desarrollo de aplicaciones, centrándose en la lógica de negocio: desarrollo, aplicación y aspectos de tiempo de ejecución

- Lograr la independencia del proveedor de componentes mediante la especificación de interfaces.

- Lograr independencia de la plataforma gracias al principio: *Write Once Run Anywhere* (WORA) y a su realización en Java

- Asegurar la compatibilidad con Java-APIs existentes, con sistemas de servidor de terceros y con protocolos de CORBA ([40]) y de servicios Web.

Para lograr estos objetivos se definen tres tipos de EJBs que se explican a continuación.

3.3.1.1 Tipos de EJBs

Existen tres tipos diferentes de EJBs:

- **EJBs de Entidad (*Entity Beans*):**

  Estos modelan conceptos de negocio como objetos persistentes asociados a datos. A partir de la versión 3.0 los Entity Beans pasan a pertenecer al API de persistencia (JPA). Se recuerda que el objetivo de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos. Se define un lenguaje propio para la persistencia, el *Java Persistence Query Language* (JPQL) y se permite la interacción con objetos simples, que en este contexto pasan a denominarse *Plain Old Java Objects* (POJOS).

- **EJB de sesión (*Session Bean*):**

  Gestionan el flujo de información del servidor. Representan procesos ejecutados en respuesta a una solicitud del cliente. Existen dos tipos distintos de EJBs de sesión:

  - Con estado (*stateful*): En estos beans de sesión un proceso de cliente involucra múltiples invocaciones, a través de las cuales se mantiene el estado, denominado *conversational state*. El estado no es persistente puesto que se pierde cuando el cliente libera el bean. Cada instancia tiene una entidad diferente.

  - Sin estado (*stateless):* Aquí un proceso cliente implica una invocación simple. Carecen de estado, así que no permiten almacenar datos específicos de cliente entre invocaciones. Todas las entidades tienen la misma instancia.

- **EJBs activados por mensajes (Message-Driven Beans):**

  Representan procesos ejecutados como respuesta a la recepción de un mensaje. Su funcionamiento es asíncrono. Utilizan el *Java Messaging System* (JMS), se suscriben a un tema (*topic*) o a una cola (*queue*) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren instanciación por parte del cliente.

3.3.1.2 Características de los EJB 3.0

La especificación 3.0 de los EJBs supuso un cambio sustancial respecto al modelo propuesto en la versión 2.1, simplificando el desarrollo de las aplicaciones y estandarizando el API de persistencia. Las principales características se resumen a continuación:

- **Especificación más simplificada**: la clase EJB es ahora una clase Java simple y llana, también conocida como POJO, y la interfaz es conocida como POJI, una simple interfaz Java. Esto aumenta la rapidez en el desarrollo de aplicaciones.

- **Anotaciones**: se utilizan anotaciones para simplificar el desarrollo de componentes. Se sigue soportando la opción del descriptor de despliegue aunque no es necesario. Las anotaciones podrán ser sobrescritas por dicho descriptor.

- **Inyección de dependencia**: el API de búsqueda y la utilización del entorno y las referencias a recursos de los EJBs se ha simplificado utilizando anotaciones. Con esto se consigue encapsular las dependencias del entorno y el acceso a JNDI.

- **Simplificación de la persistencia**: el manejo de la persistencia ahora es más sencillo con la introducción del API de persistencia de Java (JPA). La introducción del *EntityManager* ayuda mucho en este sentido.

- **Timer Service**: permite introducir un control temporal en acciones que deben ocurrir en ciertos instantes o que se deben repetir cada cierto tiempo. Se suele utilizar en Stateless Session Beans y Message-Driven Beans.

- **Interceptors y Entity Listeners:** un interceptor es una clase cuyos métodos se ejecutan cuando se llama al método de otra clase totalmente diferente. Se puede configurar para Session Beans y MDBs, pero no para los Entities.

- **Web Services**: servicios que pueden llamarse mediante los protocolos utilizados en la red por un cliente remoto. Los mensajes de petición y respuesta suelen estar basados en XML. Estos XMLs se crean bajo el estándar *Simple Object Access Protocol* (SOAP, [41]) que define reglas de serialización, empaquetado de datos y generación de mensajes. En EJB 3.0 se pueden utilizar anotaciones para su implementación.

3.3.2 Servlets

Los servlets y los JSPs son los componentes que fundamentalmente se usan en la realización de la aplicación Web de la que se encarga este PFC, ya que son apropiados para el desarrollo del nivel de presentación ([42]). Los EJBs anteriormente comentados son esenciales para la lógica de negocio, pero en nuestro caso no son necesarios porque nuestra aplicación es bastante simple, y se sigue el modelo que se ilustra en la Figura 4.

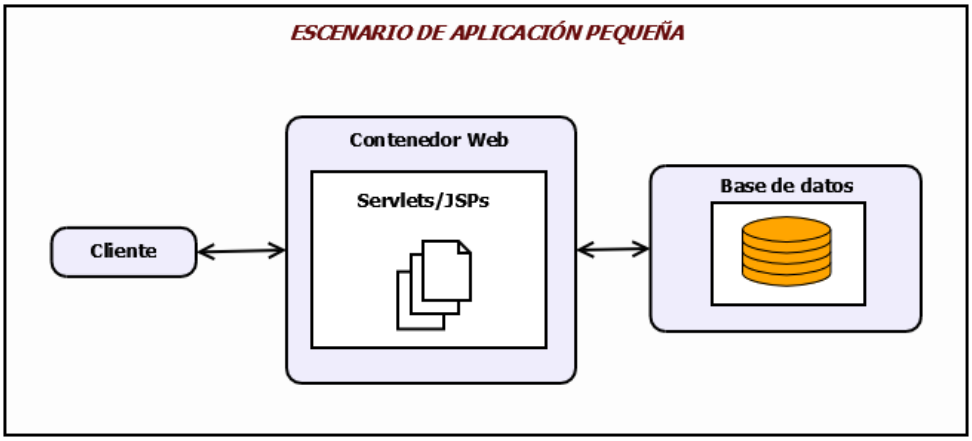


Figura 4. Arquitectura de aplicación pequeña

Un servlet no es más que una clase java usada para extender las capacidades de los servidores que albergan aplicaciones accedidas mediante un modelo de programación cliente-servidor. Su uso más común es generar páginas Web de forma dinámica a partir de los parámetros de la petición que envíe el navegador Web. Se funcionalidad es similar a los clásicos programas *Common Gateway Interface* (CGI, [43]), pero con una arquitectura de ejecución distinta, ya que están gestionados por un contenedor de servlets.

El uso de servlets proporciona numerosas ventajas como:

- Eficiencia: el servlet se instancia una única vez en el servidor, y por cada petición distinta se genera un hilo. Esto aumenta el rendimiento al disminuir los retrasos en las peticiones y además disminuye el consumo de memoria y hace el sistema más escalable. El servlet mantiene el estado entre invocaciones con lo cual no es necesario reiniciar conexiones a base de datos o red. Ejecutar una nueva petición es tan sencillo como llamar a un método.

- Utilidades para realizar las típicas tareas de servidor: los servlets proporcionan mecanismos para realizar las tareas más comunes como autenticación, gestión de errores, cookies y sesiones.

- Comunicación: los servlets poseen una manera estandarizada de comunicación con el servidor. Además permiten compartir datos entre instancias. Se pueden crear *pools* para acceder a la base de datos.

- Ventajas de Java: Al ser una tecnología Java, los servlets heredan implícitamente todas las ventajas de Java. Gran número de APIs, portabilidad entre plataformas y servidores, seguridad, orientación a objetos, gran comunidad de desarrolladores, y disponibilidad de código externo, entre otras.

3.3.2.1 El ciclo de vida de los servlets

El ciclo de vida de un servlet comienza cuando en el servidor se recibe una petición HTTP destinada a la URL del servlet en concreto. A partir de este momento se siguen los siguientes pasos:

1. Si es la primera petición, y por tanto no existen aún instancias del servlet, el contenedor Web carga la clase del servlet, crea una instancia y la inicializa llamando al método *init (ServletConfig config)*. El parámetro del método es opcional. Si no se especifica, se realiza una inicialización independiente del servidor. En caso de sí especificarlo, la inicialización es dependiente del servidor (información obtenida de descriptor de despliegue y almacenada en objeto *config*).

2. En peticiones sucesivas, el contenedor crea un hilo que llama al método *service(ServletRequest req, ServletResponse res)* de la instancia. Este método determina el tipo de petición que ha recibido y llama a un método apropiado para tratarla. Un servlet puede manejar múltiples peticiones de clientes por lo que necesita de sincronización para manejar el acceso concurrente.

3. Cuando el contenedor decide destruir el servlet, llama a su método *destroy ( ).* La destrucción del servlet puede ser debida a una decisión del administrador o a un *timeout* (demasiado tiempo inactivo). Se liberarán los recursos usados por el servlet. Si se cae el servidor Web, no se llama al método destroy, por lo que se deben guardar los datos de forma regular para mantener el estado.

3.3.2.2 API de servlets

Si se quieren usar servlets, se deben importar los paquetes *javax.servlet* y *javax.servlet.http* que contienen las clases e interfaces necesarios. Además, todos los servlets tienen que implementar el interfaz *Servlet*, que define los métodos de ciclo de vida, o bien heredar de la clase *GenericServlet* (para implementar servicios genéricos) o *HttpServlet* (para manejar servicios HTTP específicos).

Puede consultar el API en detalle en [44]. Las interfaces genéricas más importantes se listan a continuación:

- **ServletConfig:** objeto de configuración usado por el contenedor para pasar información al servlet durante la inicialización. Se recupera del descriptor de despliegue *web.xml*. La información sobre la aplicación Web a la que pertenece un servlet se almacena en este objeto.

- **ServletContext:** define un conjunto de métodos usados por el servlet para comunicarse con su contenedor (ej, obtener el tipo MIME de un fichero o repartidores de peticiones *dispatcher*) o con otros servlets de la misma aplicación Web. Hay un contexto por cada aplicación Web y por cada *Java Virtual Machine* (JVM). El contexto se obtiene a partir de la configuración mediante *ServletContext sc = Servlet.getServletConfig().getServletContext();* Se pueden almacenar y recuperar atributos mediante los métodos *setAttribute(…) y getAttribute( ).*

- **ServletRequest:** encapsula información acerca de la petición del usuario. Proporciona métodos para obtener parámetros y atributos. Por ejemplo, el método *getParameter(String name)* proporciona el valor del parámetro de nombre *name*. En un programa CGI, la decodificación de los valores de los parámetros es mucho más complicada.

- **ServletResponse:** representa la respuesta al usuario. Proporciona métodos para obtener el *Writer*, establecer el tipo de respuesta, establecer el valor de cabeceras, y más.

- **RequestDispatcher:** encapsula el reparto de peticiones. Por ejemplo, son útiles cuando es necesario redirigir de un servlet a otro servlet o JSP. Una vez obtenida la referencia al *RequestDipatcher*, el método *forward(…)* permite pasar el control al recurso que se encuentra en una URL dada. El método *include*(…) no delega el control pero incluye la salida que genera el recurso de otra URL.

3.3.2.3 Servlets HTTP

Los servlets más comunes son los Servlets HTTP, que se encargan de responder a peticiones de este protocolo. Implementar un servlet HTTP es sencillo. Lo primero es que la clase debe heredar de *javax.servlet.http.HttpServlet.* Después, se deben redefinir los métodos asociados a los distintos tipos de petición HTTP ([16]). El método *service* heredado de la superclase no se suele redefinir. Este método simplemente procesa la petición al principio para averiguar el tipo de petición y después llama al método correspondiente. Como en la gran mayoría de las ocasiones las peticiones HTTP son del tipo *GET* o *POST*, en general basta con redefinir los métodos asociados, que son *doGet (...)* y *doPost (...)* , respectivamente. En cualquier caso, el método asociado al tipo de petición *XYZ* se denominará *doXYZ (...).*

El flujo de tareas típico de un servlet HTTP es el siguiente:

1. Leer datos enviados por el usuario, típicamente a través de un formulario HTML, o desde un *applet* ([46]) o aplicación cliente.

2. Recuperar otra información de usuario embebida en la petición HTTP, como pueden ser las capacidades del navegador, *cookies* y el nombre de la máquina del cliente.

3. Generar los resultados, bien directamente, bien de manera remota vía RMI o CORBA, accediendo a una base de datos, u otras alternativas.

4. Dar formato a los resultados, típicamente en un documento HTML, pero pueden ser otros tipos como una imagen o PDF.

5. Asignar los parámetros de la respuesta HTTP, como son el tipo de documento devuelto, las cookies, parámetros de caché, y otros.

6. Por último se envía la respuesta al cliente.

El API completo de los servlets HTTP se encuentra en [45]. Aquí, se mencionará algún aspecto de las interfaces más importantes: *HttpServletRequest, HttpServletResponse* y *HttpSession.*

- **HttpServletRequest**: encapsula la información sobre la petición HTTP. Proporciona métodos para la obtención de los valores de los parámetros, manejo de las cabeceras y de la primera línea de la petición. Ejemplos:
    - *public string getParameter(String name)*: devuelve el parámetro de nombre name.
    - *String getHeader (String name):* devuelve el valor del elemento de cabecera de nombre name.
    - *Cookie[] getCookies():* devuelve todos los objetos *Cookie* que el cliente envió junto con la petición en un array de *Cookies.*

- *HttpSession getSession():* devuelve o crea una sesión asociada a la petición.
- *int getContentLength():* devuelve la cantidad en bits del cuerpo de la petición.
- *String getContentType():* devuelve el valor de la cabecera *Content-type*.
- *String getMethod():* devuelve el nombre del método de la petición (GET, POST,…) .
- *String getProtocol():* devuelve el nombre y versión del protocolo usado.

- **HttpServletResponse:** representa la respuesta HTTP y proporciona métodos que ayudan a la generación de la misma. Ejemplos:

  - *void setStatus (int sc)*: establece el código de estado de la respuesta.
  - *void sendError(int sc, String msg)*: manda el código de error y un mensaje que aparecerá en el navegador del cliente dentro de su HTML.
  - *void sendRedirect (String location):* redirección temporal al cliente con la nueva URL del parámetro.
  - *void setHeader(String name, String value):* establece la cabecera *name* al valor *value.*
  - *void setContentType (String type):* establece la cabecera Content-Type al tipo MIME ([47]) del contenido. Usado por la mayoría de servlets.
  - *void setContentLength (int len):* establece el valor de la cabecera Content-Length.
  - *void addCookie (Cookie cookie):* inserta una cookie en la cabecera Set-Cookie.

- **HttpSession:** recordamos que HTTP ([16]) es un protocolo sin estado. Esto implica que para mantener información entre peticiones (p.e, mantener información en carrito de compra en una tienda on-line, recordar login y password en un sistema con autenticación, o recordar preferencias de sitio, entre otras utilidades) se precisan mecanismos adicionales. Los principales mecanismos son:

  - Cookies: las cookies son piezas pequeñas de información enviadas del servidor al cliente en respuestas HTTP y retornadas del cliente al servidor en sucesivas peticiones. Una cookie es por tanto un medio para que el servidor almacene información en el cliente. Algunos usuarios las desactivan porque pueden suponer un problema de privacidad.

  - URL-rewriting: se debe añadir la información de sesión a todas las URLs que refieren al sitio Web propio. Esto implica que no se puede usar páginas estáticas que contengan tales URLs.

  - Hidden form fields: este método es un poco más tedioso, y obliga a que todas las páginas deben ser resultado de formularios previos.

La interfaz HttpSession encapsula las funciones necesarias para crear sesiones entre el cliente y el servidor HTTP, que persisten a través de distintas peticiones. Además, permite ver y manipular información de una sesión, tal como el identificador de sesión, el momento de creación o el máximo tiempo inactivo, y también permite vincular objetos a sesiones, permitiendo que la información de usuario persista a través de varias conexiones. Por detrás, el mecanismo de control de sesión suele usar cookies, o si están deshabilitadas, reescritura de URLs. Se puede obtener un objeto *HttpSesion* invocando al método *getSession ()* de *HttpServletRequest.* Como ya indicamos, este método devuelve el objeto de sesión asociado a la petición, o bien crea una nueva sesión si no existiera. Los principales métodos que soporta este objeto son:

- *void setAttribute(String name, Object value):* asocia el objeto *value* a la sesión, identificándolo como un atributo de nombre *name.*
- *Object getAttribute(String name):* devuelve el atributo de sesión con nombre *name.*
- *void removeAttribute(String name):* elimina de la sesión el atributo de nombre *name.*
- *String getId():* devuelve el identificador de sesión.
- *long getCreationTime():* devuelve el momento de creación de la sesión.
- *void setMaxInactiveInterval(int interval):* establece el máximo tiempo inactivo permitido. Si se supera este período la sesión se destruye automáticamente.
- *int getMaxInactiveInterval():* devuelve el máximo tiempo inactivo configurado.
- *void invalidate():* destruye la sesión.

3.3.3 Java Server Pages (JSPs)

Los servlets generan siempre toda la página de manera dinámica. Sin embargo, en muchas ocasiones la mayor parte del contenido es estático y sólo una pequeña parte es variable. En estos casos, escribir un servlet puede resultar un poco ineficiente. La solución a este problema la proporcionan las *Java Server Pages* (JSPs). Esta tecnología permite entremezclar contenido HTML estático con contenido dinámico generado con distintas etiquetas o código Java, y está ampliamente soportada por plataformas y servidores Web. En última instancia, un servlet y una JSP son equivalentes, puesto que las JSP son compiladas por el servidor a un servlet la primera vez que se usan o se despliegan. La diferencia pues, reside más bien en el enfoque de la programación o la manera de escribir el código.

3.3.3.1 Procesamiento JSP

El procesamiento que sigue una página JSP dentro del servidor cuando se recibe una petición a su URL por primera vez consta de los siguientes pasos:

1. Se traduce la página JSP a un servlet (código Java)
2. Se compila el servlet a bytecode (*.class)*
3. Se carga la clase del servlet
4. Se crea la instancia del servlet
5. Se llama al método de inicialización *jspInit*
6. Se llama al método *_jspService* para atender peticiones
7. Se llama al método *jspDestroy* si se quiere destruir la instancia

En las sucesivas peticiones, al estar ya inicializado, se empezaría directamente en el punto 6. A continuación, se analizan brevemente los distintos elementos de la tecnología JSP. Un tutorial más completo está disponible en [48].

3.3.3.2 Variables implícitas

Cuando se escriben páginas JSP hay una serie de variables que se pueden usar como si ya estuvieran definidas. Son las llamadas variables implícitas. Las más usadas son:

- request: representa el objeto *HttpServletRequest*
- response: representa el objeto *HttpServletResponse*
- session: representa el objeto *HttpSession* asociado a la petición
- out: representa el objeto *PrintWriter* usado para enviar la salida al cliente
- page: sinónimo de *this*
- application: representa el objeto *ServletContext*
- config: representa el objeto *ServletConfig*

El disponer de estas variables aporta bastante agilidad a la hora de escribir el código, pues no se tienen que declarar ni llamar a ninguna función para obtenerlas, sino que se usan directamente, llamando a sus métodos cuando sea preciso.

3.3.3.3 Instrucciones JSP

Las páginas JSP permiten introducir distintos tipos de instrucciones embebidas que agilizan la escritura y configuración de las mismas:

- **Guiones o scripts**

  Especifican código Java que tras la traducción forma parte del servlet. Existen tres tipos de scripts diferentes:

  - Expresiones: son evaluadas y convertidas a *String* y su resultado se incluye en la salida. Su sintaxis es *<%= expression %>*.
    Ej: * Your session Id: <%= session.getId() %> *

  - Scriptlets: son bloques de código java de una página JSP que se insertan en el método *_jspService* del servlet. Su sintaxis es *<% code %>*.

Generalmente se ocupan de tareas que no pueden realizarse mediante expresiones, como generar cabeceras de respuesta, ejecutar un código que contenga bucles o actualizar una base de datos.
Ej: *<% response.setContentType("text/plain"); %>*

- Declaraciones: es código de inicialización, se insertará en la clase del servlet pero fuera de métodos existentes. Su sintaxis es *<%! code %>*. Se suelen utilizar para definir métodos o campos que luegon se utilizarán en expresiones o scriptlets. Ej:
  <%! private int accessCount = 0; %>
  <h2>Accesses to page since server reboot:
  <%= ++accessCount%></h2>

- **Directivas**

Las directivas controlan la estructura general del servlet que se genera de la página JSP. La sintaxis de las directicas es *<%@ directive attribute1="value1" ... attributeN="valueN" %>*. Existen tres tipos de directivas:

- page: controla la estructura del servlet importando clases, adaptando la superclase o configurando el tipo de contenido, entre otras acciones
  Ej: *<%@ page import="java.util.*, java.io.*" %>*

- include: permite insertar el contenido de otros ficheros (HTML, JSP) en el servlet en el momento de la traducción de JSP a servlet.
  Ej: *<%@ include file="URL relative" %>*

- taglib: extiende la funcionalidad de JSP, permitiendo definir etiquetas de marcado personalizadas (*custom tags*). El desarrollador define la interpretación de cada etiqueta, sus atributos y su cuerpo. Las etiquetas se pueden agrupar en librerías de etiquetas. Existe un conjunto de librerías predefinidas para las funcionalidades más comunes, como los aspectos de formato, procesamiento XML o acceso a base de datos. Juntas componen la *JSP Standard Tag Library* (JSTL). Se define además un lenguaje propio denominado *Expresion Language* (EL) pensado para extender la funcionalidad.

- **Acciones**

Son etiquetas embebidas en una página JSP que se interpretan en tiempo de ejecución. Controlan la ejecución del motor de JSPs. Su sintaxis es *< jsp:acción atributos >*. Las principales son *jsp:include*, para incluir contenido de otra url después de la traducción al servlet, *jsp:forward*, para pasar el control a otra url y *jsp:param*, para incluir parámetros.

Ej: *<jsp:forward page="list.jsp">*
<jsp:param name="order" value="date" />
</jsp:forward>

Por último, apuntar que los comentarios dentro de las páginas JSP se escriben con la sintaxis <%-- *comentario* --%>. Comentar también que existe una sintaxis alternativa para JSP para acoplarse con XML ([49]).

3.3.4 Java Beans

Los *Java Beans* son sencillamente clases java que ayudan a una separación más fuerte entre contenido y presentación. En lugar de tener grandes cantidades de código embebido en páginas JSP, se encapsula la información en objetos beneficiando la reusabilidad y modularidad. Además, facilita la compartición de objetos entre páginas y servlets y pueden simplificar el proceso de lectura de parámetros de las peticiones. Están perfectamente integrados con la tecnología JSP, que pueden acceder y modificarlos de manera sencilla mediante las acciones *jsp:useBean, jsp:setProperty* y *jsp:getProperty*. Las *properties* se refieren a los atributos del Bean, que deben declararse como privados. Asimismo, deben definirse los métodos de acceso y modificación de los atributos (*get* y *set)*. Los beans pueden mandar notificaciones de cambios en sus propiedades.

Puede especificarse el ámbito de validez de los Beans mediante el atributo *scope* en el momento de la creación. Este puede tomar cuatro valores: *page* (sólo en la misma página), *request* (válido dentro de la misma petición), *sesión* (accesible en todas la peticiones de una misma sesión) y *application* (ámbito de toda la aplicación, puede accederse a través de la variable predefinida del mismo nombre).

3.4 Integración entre Servlets y JSPs

Hemos visto que las JSP son más adecuadas que los servlets para páginas de presentación con mucho contenido HTML estático. Por tanto, una solución sólo JSP puede ser adecuada la salida está basada principalmente en caracteres y el formato/layout es prácticamente fijo. Los servlets, por su lado, son más eficaces en procesar datos y generar contenido dinámico. Una aplicación de sólo servlets puede ser efectiva si la salida es un tipo binario o no existe salida, o bien el layout de la página es altamente variable. La mayoría de las aplicaciones empresariales no se encuentra en estos extremos, y por tanto una solución de compromiso consiste en combinar el uso de servlets y JSP.

Una solución bastante común consiste en procesar la petición inicial con un servlet (controlador) que procesa parcialmente los datos y configura los beans (modelo). Luego pasa el control a distintas páginas JSP (vista) que muestran la interfaz final al usuario dependiendo de las circunstancias. Este patrón de diseño es muy habitual en aplicaciones Web y recibe el nombre de patrón Modelo-Vista-Controlador (MVC, [50]).

El *modelo* o lógica de negocio es la parte que se encarga del manejo de la información. Interactúa con la base de datos y describe funciones de procesamiento. Representa el estado de la aplicación. La *vista* recibe los datos pertinentes y se encarga de la interfaz gráfica de cara al cliente. Por último, el *controlador*, se ocupa de manejar el flujo del programa. Además, establece una interfaz o punto de unión entre las otras dos partes que permite su desacoplo, las independiza. Esto favorece la reusabilidad y

modularidad, permitiendo, por ejemplo, que la lógica de negocio y el nivel de presentación de una misma aplicación Web sean desarrollados por equipos distintos. Un esquema básico del patrón MVC se ilustra en la Figura 5.

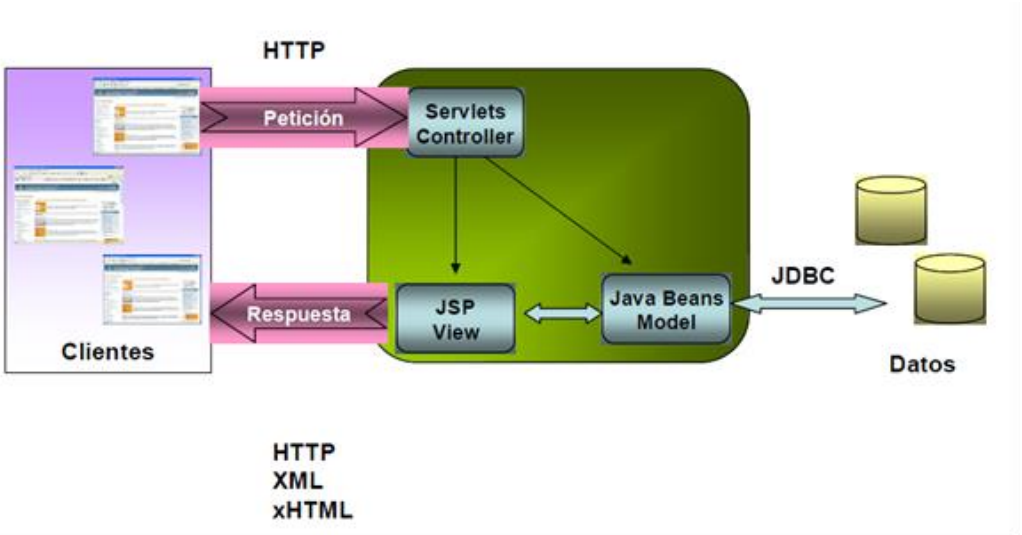


Figura 5. Patrón MVC ([51])

Capítulo 4

El servidor Apache Tomcat

4.1 Introducción

Apache Tomcat ([9]) es una herramienta que aúna las funcionalidades de un servidor Web con las de un contenedor de servlets y JSPs de la especificación Java EE. También llamado Jakarta Tomcat o simplemente Tomcat, es desarrollado y mantenido por miembros de la Apache Software Foundation ([53]) y voluntarios independientes bajo el nombre de Proyecto Jakarta. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software License.

Es necesario matizar que Tomcat es simplemente un servidor Web con soporte de servlets y JSP, pero no llega a ser un servidor de aplicaciones completo como pueden ser JBoss AS, JonAS o GlassFish.

Apache Tomcat incluye herramientas para su gestión y configuración, pero también puede ser configurado mediante la edición de ficheros XML. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java (JVM).

4.1.1 Historia y desarrollo de las distintas versiones

Tomcat empezó siendo una implementación de la especificación de los servlets comenzada por James Duncan Davidson ([54]), que trabajaba como arquitecto de software en Sun Microsystems y que posteriormente ayudó a hacer al proyecto de código abierto y en su donación a la Apache Software Foundation.

Duncan Davidson inicialmente esperaba que el proyecto se convirtiese en software de código abierto y dado que la mayoría de los proyectos de este tipo tienen libros de O'Reilly asociados con un animal en la portada, quiso ponerle al proyecto nombre de animal. Eligió Tomcat (gato), pretendiendo representar la capacidad de cuidarse por sí mismo, de ser independiente.

Figura 6. Logotipo de Apache Tomcat ([9])

Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 7.x, que soportan las últimas especificaciones de servlets y JSPs. Se incluye a continuación una lista de las distintas versiones con algunas de sus características.

- **Tomcat 3.x (distribución inicial)**
    - Implementado a partir de las especificaciones Servlet 2.2 y JSP 1.1
    - Recarga de servlets
    - Funciones básicas HTTP
- **Tomcat 4.x**
    - Implementado a partir de las especificaciones Servlet 2.3 y JSP 1.2
    - Contenedor de servlets rediseñado como *Catalina*
    - Motor JSP rediseñado con *Jasper*
    - Conector *Coyote*
    - *Java Management Extensions* (JMX), JSP y administración basada en *Struts*
- **Tomcat 5.x**
    - Implementado a partir de las especificaciones Servlet 2.4 y JSP 2.0
    - Recolección de basura reducida
    - Capa envolvente nativa para Windows y Unix para la integración de las plataformas
    - Análisis rápido JSP
- **Tomcat 6.x**
    - Implementado de Servlet 2.5 y JSP 2.1
    - Soporte para *Unified Expression Language* 2.1
    - Diseñado para funcionar en Java SE 5.0 y posteriores
    - Soporte para *Comet* a través de la interfaz *CometProcessor*

- **Tomcat 7.x**

    - Implementado de Servlet 3.0 JSP 2.2 y EL 2.2
    - Mejoras para detectar y prevenir "fugas de memoria" en las aplicaciones
    - Limpieza interna de código
    - Soporte para la inclusión de contenidos externos directamente en una aplicación Web

En el proyecto se ha utilizado la versión 7. Puede consultar los detalles de instalación y configuración en el Apéndice B.3.

4.2 Componentes

A partir de la versión 4, Tomcat fue rediseñado a través de componentes que proporcionan las distintas funcionalidades requeridas:

- **Catalina:** es el motor de servlets de Tomcat, que implementa las especificaciones de Oracle para servlets y JSPs.

- **Coyote:** es el conector HTTP que usa Tomcat y soporta las funcionalidades concernientes al protocolo HTTP 1.1. Escucha a la espera de conexiones entrantes a un puerto TCP específico del servidor y reenvía las peticiones al motor de Tomcat para las procese. Luego devuelve las respuestas generadas a los clientes correspondientes.

- **Jasper:** es el motor de JSP de Tomcat. Transforma los archivos JSP a servlets que pueden ser manejados por Catalina. En tiempo de ejecución, Jasper detecta cambios en los ficheros JSP y los recompila. A partir de la versión 5 de Tomcat, se desarrolla Jasper 2, que añade importantes características en el manejo y compilación de JSPs.

- **Cluster:** este componente ha sido añadido para gestionar aplicaciones de gran tamaño. Se usa principalmente para conseguir balanceo de carga, que puede ser logrado mediante varias técnicas. El soporte para Clustering requiere al menos de la versión 1.5 del JDK o superior.

4.3 Estructura de directorios y ficheros de configuración

La jerarquía de directorios de instalación de Tomcat incluye:

- **bin:** arranque, cierre, y otros scripts y ejecutables
- **lib:** aquí se incluyen distintas librerías necesarias y podemos añadir las propias para funciones extra
- **common:** clases comunes que pueden utilizar Catalina y las aplicaciones Web
- **conf:** ficheros XML y los correspondientes DTD para la configuración de Tomcat
- **logs:** *ficheros de bitácora* de Catalina y de las aplicaciones
- **server:** clases utilizadas solamente por Catalina
- **shared:** clases compartidas por todas las aplicaciones Web
- **temp:** almacenamiento temporal para la máquina Java
- **webapps:** directorio que contiene las aplicaciones Web
- **work:** almacenamiento temporal de ficheros y directorios

Los principales ficheros de gestión y configuración son los siguientes:

- **Dentro del directorio *conf***
  - server.xml: archivo principal de configuración
  - tomcat-users.xml: permite crear usuarios/contraseñas y roles
  - web.xml: valores por defecto para todas las aplicaciones
  - catalina.properties: para modificar la estructura de directorios
  - catalina.policy: políticas de seguridad

- **Dentro del directorio *bin***
  - catalina.sh: parámetros del arranque
  - startup.sh: script de arranque
  - shutdown.sh: script de parada
  - version.sh: datos de la versión

4.4 Configuración de una aplicación Web en Apache Tomcat

Para que Tomcat pueda reconocer los componentes de nuestra aplicación y ejecutarlos cuando sean requeridos, es necesario instalar estos componentes en el contenedor. Este proceso recibe el nombre de despliegue de la aplicación. Se deben tener en cuenta dos aspectos básicos para la realización del mismo: la estructura de directorios que la aplicación debe respetar y un fichero especial denominado descriptor de despliegue.

➢ **Estructura de directorios de las aplicaciones**

Para cada aplicación Web que se quiera instalar en Tomcat, se debe crear un nuevo directorio dentro del directorio *webapps* de la instalación de Tomcat. Este nuevo directorio será específico para la nueva aplicación y define su contexto. Habitualmente recibe el nombre del nombre de la aplicación. A partir de aquí la estructura es la siguiente:

- **Directorio *webapps/NombreApp***: representa la parte pública de la aplicación, es decir, los ficheros que se pongan en este directorio podrán ser accedidos directamente a través de la Web por los usuarios. Es por ello por lo que se suelen ubicar aquí ficheros tales como páginas HTML estáticas, imágenes, hojas de estilo CSS y páginas JSP.

    - **Subdirectorio *webapps/NombreApp/WEB-INF*:** representa la parte privada de la aplicación, que no puede ser accedida directamente por los clientes. Aquí se debe ubicar un fichero *web.xml*, que será el descriptor de despliegue que configura la aplicación.

        - **Subdirectorio *webapps/NombreApp/WEB-INF/classes*:** aquí van los ficheros compilados tales como servlets o beans, de las clases utilizadas por la aplicación Web.

        - **Subdirectorio *webapps/NombreApp/WEB-INF/lib*:** en él se colocan otras bibliotecas de clases adicionales (comprimidas con *jar*) que utilice tu aplicación.

    - **Resto de subdirectorios:** también son públicos, así que se puede ubicar en ellos ficheros estáticos y JSP. Se pueden crear para organizar mejor los ficheros.

Para exportar aplicaciones Web a otros servidores, esta estructura puede ser comprimida, usando jar, dando lugar a un fichero WAR (*Web ARchive*), que se suele almacenar con la extensión war, por ejemplo, *NombreApp.war*. Tales ficheros son compatibles con cualquier plataforma de ejecución de servlets y JSPs.

➢ **Descriptor de despliegue**

Este fichero provee al contenedor información relevante de configuración de la aplicación. Se denomina *web.xml* y se debe incluir en el directorio *WEB-INF*. Contiene la descripción de la aplicación Web: permite declarar servlets, asignarles parámetros de inicio, declarar alias y filtros y configurar sesiones, entre otras funcionalidades.

Como ejemplo, se incluye a continuación la declaración de un servlet de nombre *hola* correspondiente a una clase *paquete.HolaMundo*. Posteriormente, se realiza el mapeo de dicho servlet a la URL relativa */servlet/hola*. Como resultado, este servlet será accedido al invocar la dirección *http://domain:port/NombreApp/servlet/hola*.

```
<servlet>
   <servlet-name>hello</servlet-name>
   <servlet-class>paquete.HolaMundo</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>hello</servlet-name>
   <url-pattern>/servlet/hola</url-pattern>
</servlet-mapping>
```

Se pueden añadir parámetros de inicialización con la etiqueta <*init-param*>, o especificar el fichero de bienvenida con <*welcome-file-list*>. La descripción de todas las etiquetas admitidas con su función puede encontrarse dentro de la documentación de Oracle en [55].

Capítulo 5

Diseño de la aplicación

5.1 Introducción

Después de hacer un repaso a la tecnología Java EE y al servidor Apache Tomcat, que permitirá desplegar nuestra aplicación, vamos a centrarnos en el diseño de la misma. Se recuerda que el problema que se quiere resolver es la realización de una interfaz Web para la estrategia de hitos descrita en los primeros capítulos de esta memoria. Se comenzará dando una visión global de la funcionalidad que pretende ofrecer la aplicación mediante los distintos casos de uso que se generan. Luego se ilustrarán los componentes necesarios y la interacción entre los mismos para lograr los objetivos.

5.2 Casos de uso

El sistema dispone de dos tipos de usuarios o actores principales: alumnos y profesores. También existe el rol de administrador, que puede considerarse como una extensión del rol profesor, con privilegios extra para configurar la aplicación. Cada rol permite al usuario el acceso a ciertas tareas. Esto lo vamos a esquematizar mediante los diagramas de casos de uso. En aras de claridad, vamos a realizar diagramas distintos para el perfil de alumno (Figura 7) y profesor/administrador (Figura 8).

Como se aprecia en el esquema, la función del alumno en el sistema es sencilla. La idea es que su participación sea ágil y ligera para no incurrir en una sobrecarga que pueda desmotivar el uso de la plataforma. Podrá acceder a una encuesta directamente mediante el enlace (*link)* de la misma, que le será facilitado por el profesor. Allí podrá ver la información de los alumnos que ya la hayan completado, e introducir/actualizar su propia información. También tiene la opción de acceder a una pantalla que lista las encuestas en las que está activo, y acceder a cualquiera de ellas con un click. Previamente a cualquiera de estas acciones, se requerirá la autenticación en el sistema mediante un *login* y *password,* para asegurar la veracidad de la identidad de los participantes (sustituye a la firma en las encuestas de papel).

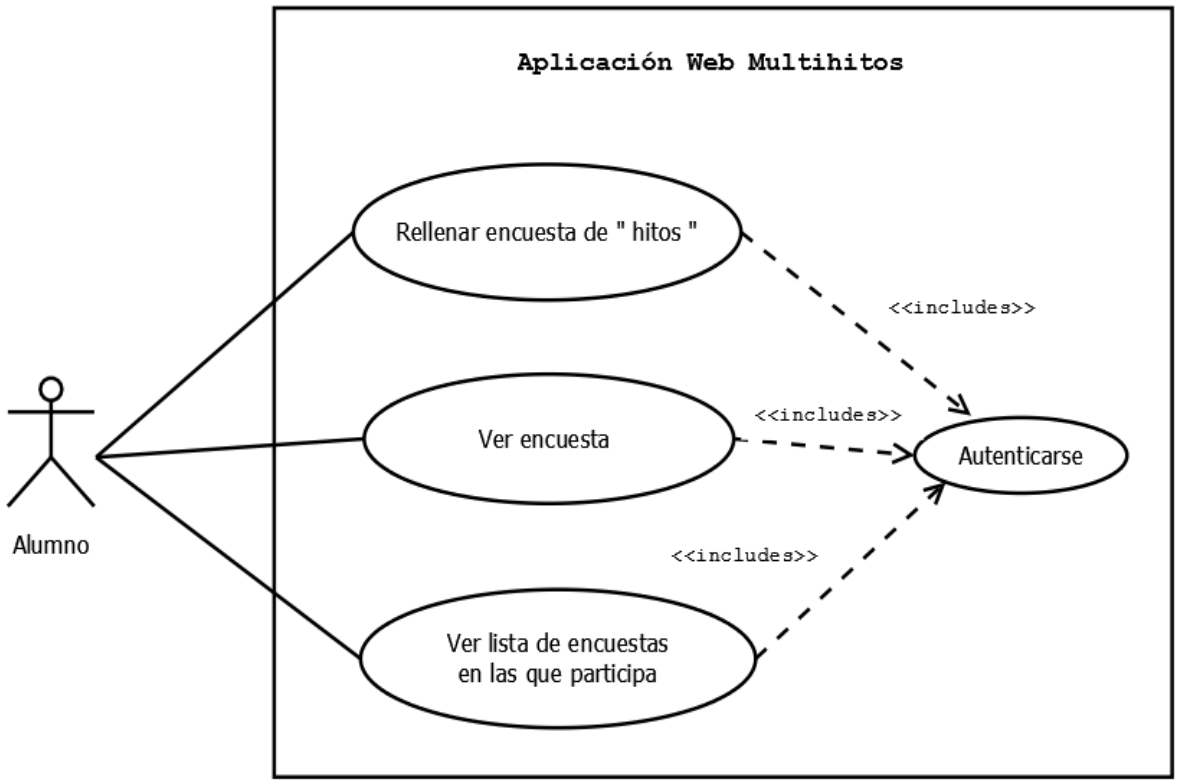


Figura 7. Diagrama de casos de uso para el perfil de alumno

El perfil de profesor es un poco más complejo. Podrá crear encuestas con distintas opciones de configuración, que posteriormente pasará a los alumnos para que las rellenen. También puede obtener un listado con las encuestas que gestiona, y acceder a las mismas o eliminarlas del sistema. El acceso a la encuesta en este caso es significativamente distinto al acceso del alumno, pues permite una edición completa de la misma (añadir o eliminar campos y añadir, modificar o eliminar los registros de los participantes) además del acceso a las opciones de configuración de la tabla (fechas de validez, descripción, permitir a otros profesores la gestión de la tabla). De nuevo, se pedirá una autenticación inicial antes de poder entrar el sistema para realizar cualquiera de estas acciones.

El administrador o *root* utiliza el sistema como otro profesor, con la diferencia de que tiene acceso a todas las encuestas del sistema independientemente de que las haya o no creado, o le hayan otorgado permisos. Además, tiene la opción adicional de modificar la configuración global del sistema, como los datos necesarios para la autenticación con LDAP.

La idea general se resume en la Figura 8:

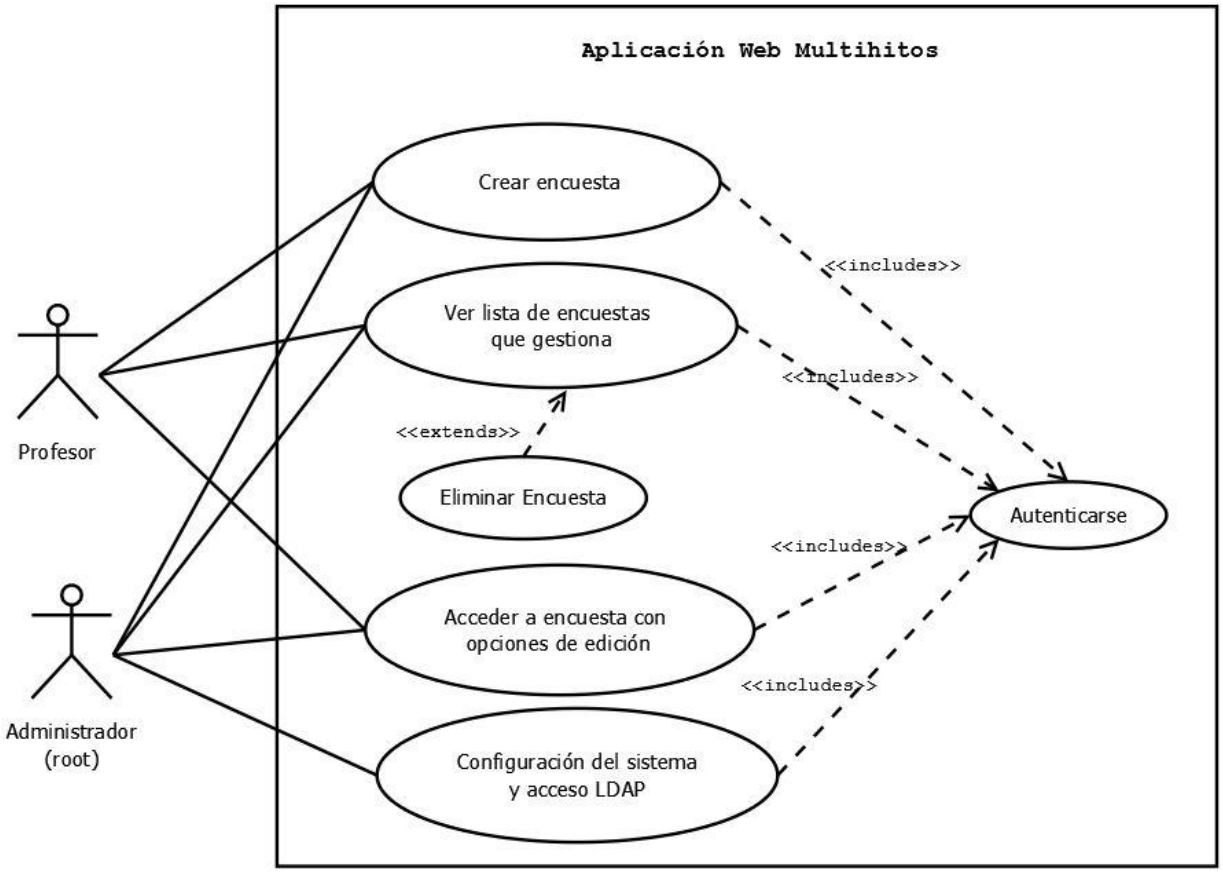


Figura 8. Diagrama de casos de uso para el perfil de profesor y administrador

5.3 Despliegue de la aplicación y componentes

La aplicación precisará de una serie de componentes que deben relacionarse entre sí para obtener el resultado esperado. Debemos delimitar estos componentes y especificar el ámbito en el que van a ser desplegados. En este caso, se puede considerar nuestra herramienta como una aplicación Web convencional que sigue el paradigma cliente-servidor, para las cuales existen diversos patrones de diseño ya establecidos que ayudan al desarrollo de las mismas. Se ha optado por el típico diseño MVC (Figura 5), que independiza las tareas relacionadas con la interfaz, el control, y los datos de la aplicación. El resultado es el diseño esquemático que se muestra en la Figura 9. En ella se aprecian los distintos bloques o componentes que se distinguen.

El primer bloque es el del usuario. El usuario es el alumno, profesor, o profesor administrador que va a realizar el papel de *cliente* en el sistema. Como la interfaz propuesta es una interfaz Web, el medio de acceso es sencillamente un navegador convencional, con el que se accederá a la URL del servidor de la aplicación.

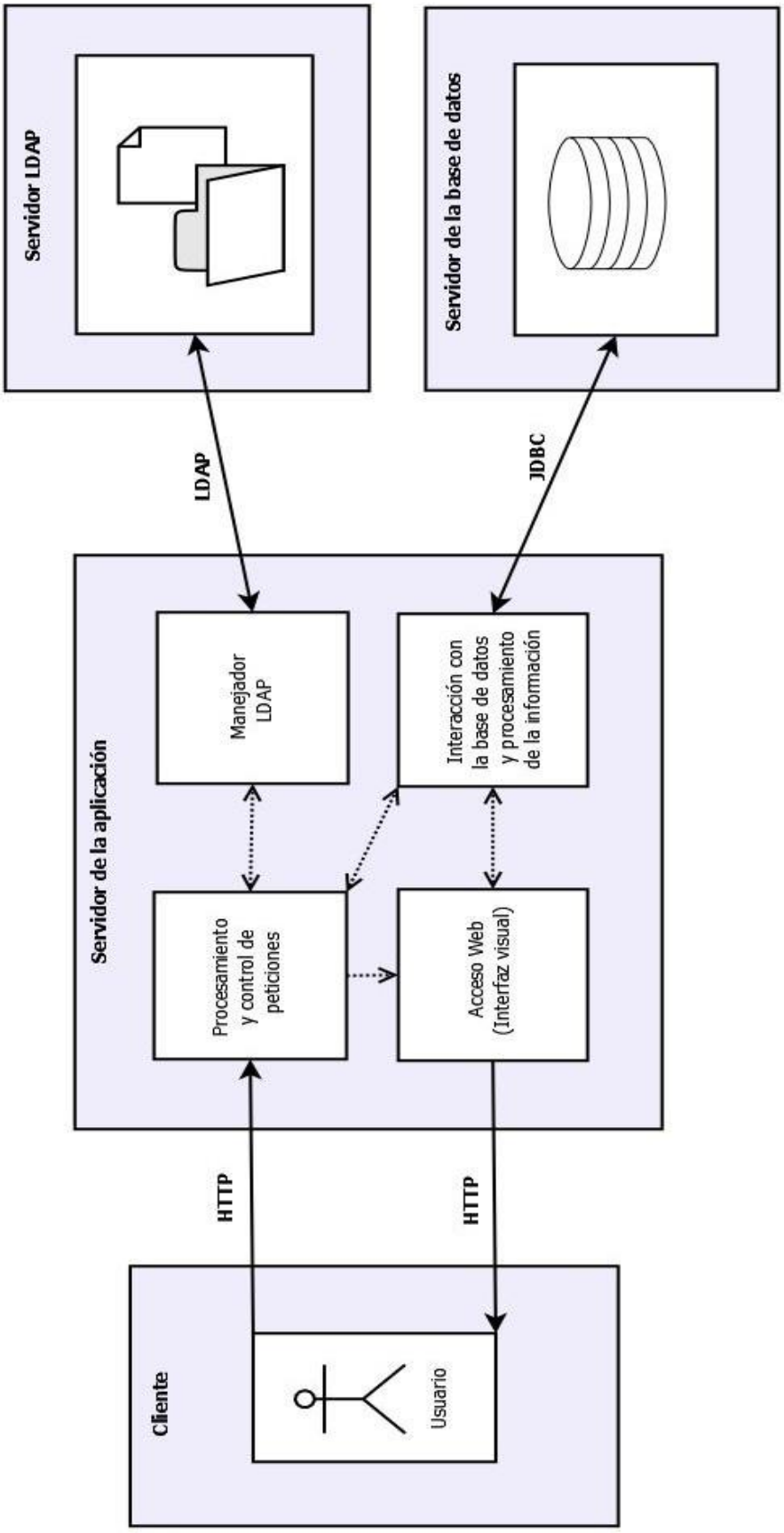


Figura 9. Despliegue de la aplicación

En el *servidor de la aplicación* se diferencian cuatro componentes separados:

- Procesamiento y control de peticiones: este bloque se encarga de procesar las órdenes que recibe del *cliente* mediante las peticiones. Trata cada petición de la manera oportuna, procesando los distintos parámetros de las mismas. Interacciona con el componente *manejador LDAP* para el control de autenticación, y también con el componente *interacción con la DB* si la petición precisa consultar alguna información del sistema. Indica al componente *Acceso Web* qué interfaz corresponde mostrar al usuario

- Acceso Web: se encarga del nivel de presentación. Genera las respuestas, en nuestro caso tipo XHTML, que se envían de vuelta al cliente para que se muestren en su navegador.

- Interacción con la DB: este componente recibe y gestiona todas las peticiones a la base de datos. Realiza el procesado de la información necesario e interacciona con la base de datos mediante el uso de la API *Java DataBase Connectivity* (JDBC, [56]). Esta API, además, permite que el código del programa sea independiente del sistema gestor de la base de datos. De este modo, si en algún momento se produce un cambio del SGBD en el sistema, bastará con sustituir el fichero del driver asociado al sistema gestor, y no será necesario modificar el código del programa.

- Manejador LDAP: Aquí se encapsula la funcionalidad requerida para el mecanismo de autenticación en el sistema mediante el protocolo LDAP. Proporciona al componente de control la interfaz necesaria para configurar los parámetros básicos y realizar búsquedas.

Los dos componentes restantes son el *servidor de la base de datos* y el *servidor LDAP*. Su función es el almacenamiento de la información relevante en cada caso, proporcionado métodos para la modificación y recuperación pertinente de dicha información.

Es necesario matizar que la división en componentes aquí propuesta se trata de una división lógica y no física. Es decir, puede que varios de los componentes o incluso todos se ubiquen en el mismo equipo físico. De hecho, para el desarrollo y pruebas de la aplicación, se instalaron en la misma máquina virtual todos los componentes excepto el servidor LDAP, ubicado en la universidad. No obstante, lo más normal en la realidad es encontrarse con un entorno mucho más distribuido donde el cliente, el servidor de la aplicación, el servidor de la base de datos y el servidor LDAP están en máquinas distintas y lugares distintos.

5.4 Interacción entre los componentes

Una vez introducidos los componentes fundamentales para la implantación de la aplicación, se describirá cómo colaboran entre ellos para conseguir que el sistema funcione. Se utilizará el diagrama de secuencia como herramienta gráfica para ilustrar el funcionamiento de los principales casos de uso del sistema.

En la Figura 10, se muestra la interacción típica entre los componentes para la tarea de *rellenar una encuesta* por parte de un alumno. Se aprecia el proceso de autenticación con LDAP requerido para acceder al sistema, la consulta a la base de datos para obtener los datos de la encuesta, la introducción de los datos correspondientes por el alumno que actualizan la base de datos, y por último el cierre de sesión en el sistema. En el caso de acceso a una encuesta por un profesor el proceso es equivalente, con la diferencia de que una vez cargada la encuesta, dispone de muchas más opciones a realizar sobre la misma. Si alguno de los pasos fallara, (p.ej., autenticación fallida, o introducción de datos con formato incorrecto), se mostrará un mensaje de error y se vuelve a repetir el paso.

En el resto de figuras, que ilustran las tareas de *ver lista de encuestas*, *crear tabla*, y *configurar sistema* la filosofía es muy similar. El usuario hace peticiones contra el bloque de *procesamiento y control de peticiones*. Si es la primera petición de una sesión, se debe pasar el control al *manejador LDAP* para el proceso de autenticación. En las sucesivas peticiones, se pasa el control al componente *acceso Web* para que devuelva una interfaz, o bien a *interacción con la DB,* si hay que consultar o modificar información.

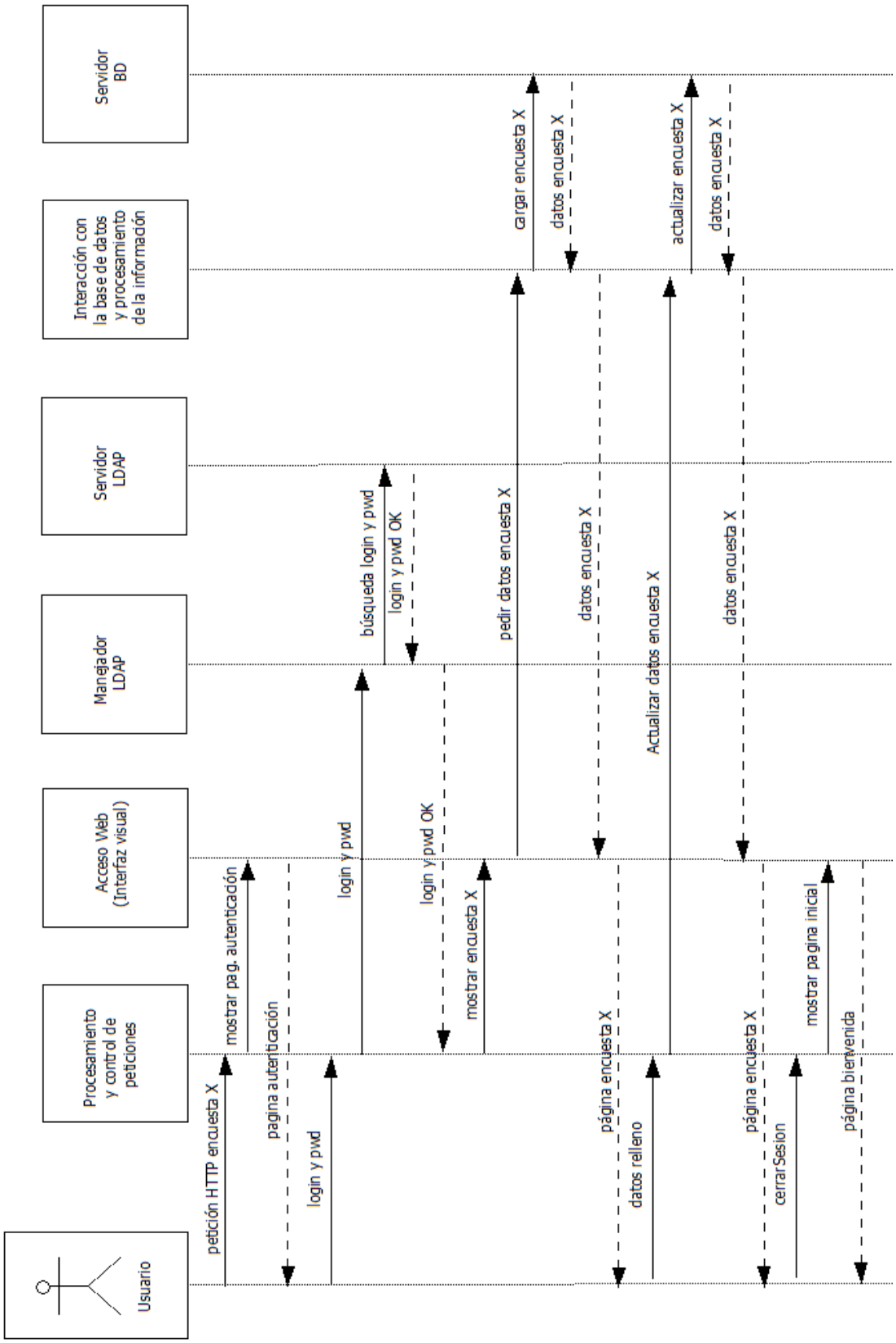


Figura 10. Diagrama de secuencia de la acción “rellenar una encuesta”

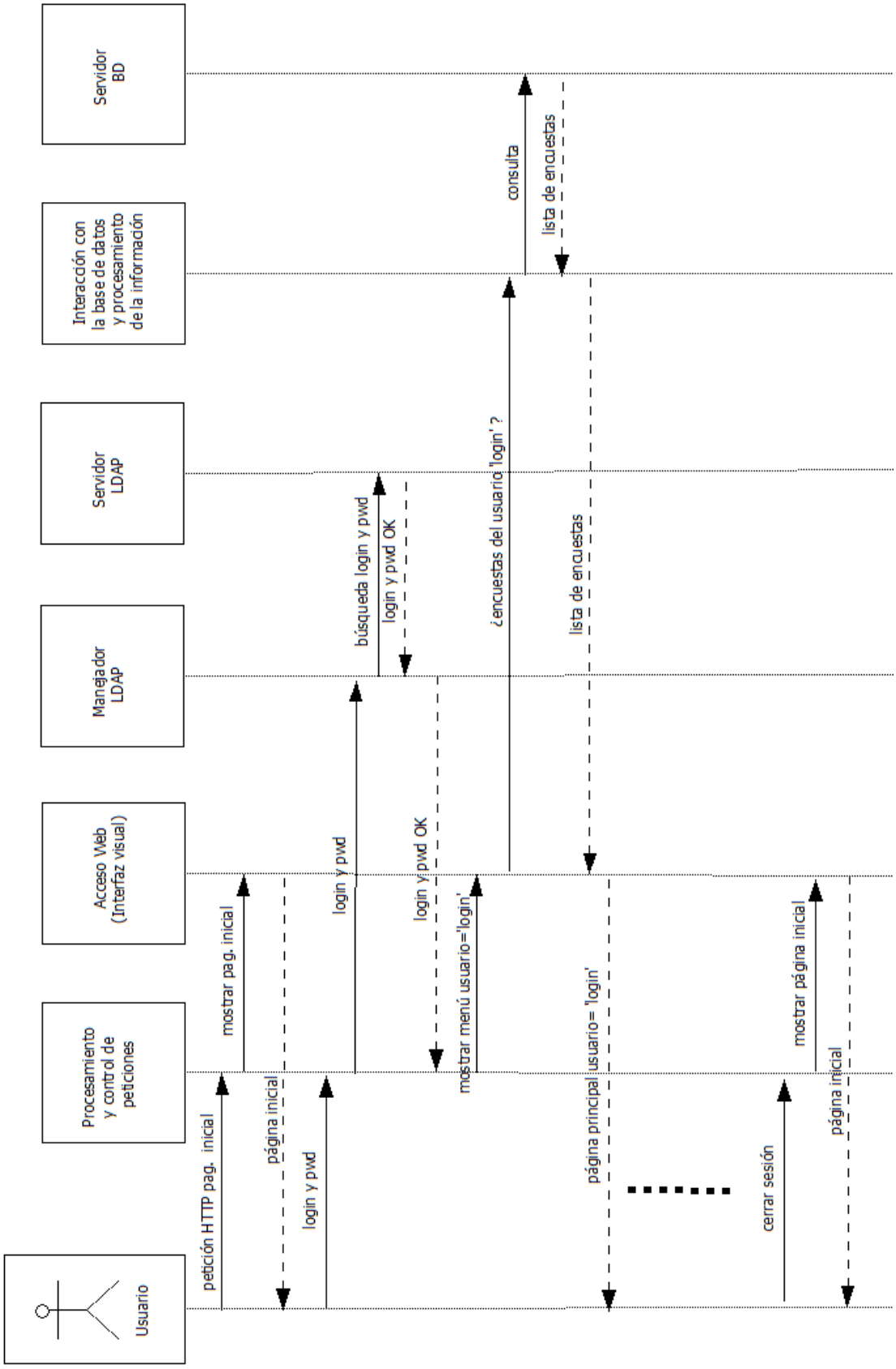


Figura 11. Diagrama de secuencia de la acción “ver lista de encuestas”

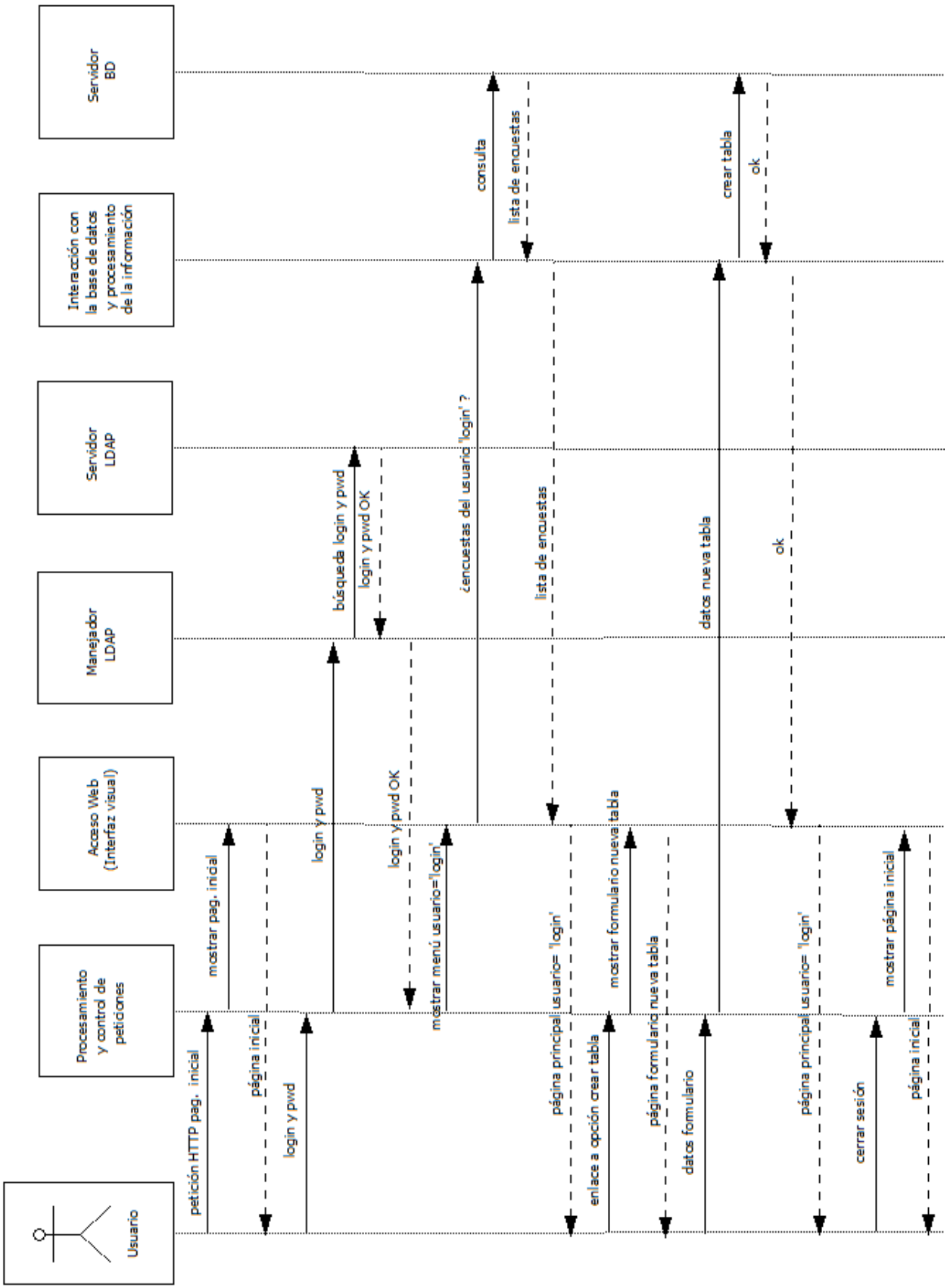


Figura 12. Diagrama de secuencia de la acción crear encuesta

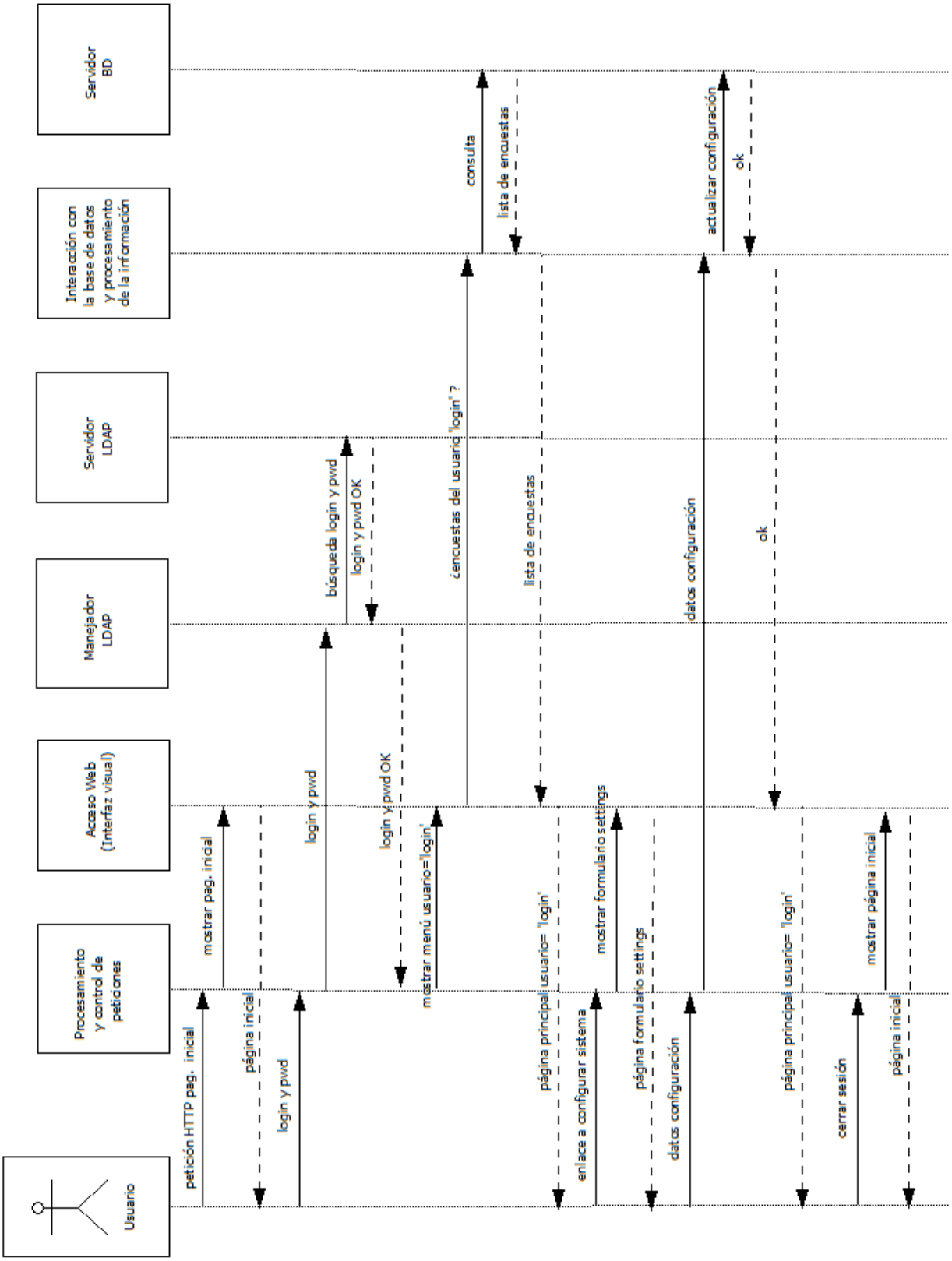


Figura 13. Diagrama de secuencia de la acción configurar sistema

5.5 Diseño de la base de datos

Un paso importante previo a la implementación en los sistemas en los que se precisa de una base de datos es el diseño de la misma. Se necesita conocer cómo va a estar estructurada la información para sus consultas y modificaciones. Para la aplicación de este PFC, la base de datos es sencilla. Por la estructura de las propias encuestas, es natural establecer una correspondencia directa entre encuesta y tabla en la base de datos. Es decir, por cada encuesta que se cree, se creará una tabla asociada en la base de datos relacional.

Por otra parte, se precisa de otros elementos adicionales para la organización y gestión del sistema. En concreto:

- Un índice de las encuestas existentes en el sistema: por cada encuesta que se cree, se almacenará su nombre e información de configuración asociada: nombre del profesor que la ha creado, fechas de validez, nombre de los profesores a los que se permite el acceso a la tabla, campo para indicar si debe ser visible o no para los alumnos una vez pasado el plazo límite, y una breve descripción textual de la misma.

- Tabla de administradores: en esta tabla se almacenarán los *logins* de aquellos profesores que también tienen permisos de administración (*root*). Los administradores tienen acceso a todas las encuestas del sistema.

- Tabla de configuración LDAP: para dar mayor flexibilidad al sistema, los datos de conexión LDAP van a ser configurables, lo cual permitirá adaptar el sistema a distintos entornos, por ejemplo, a otra universidad. Dependiendo del entorno, podremos tener varios servidores de autenticación. En nuestro caso, se habilitará un LDAP para profesores y otro para alumnos, aunque configuramos ambos con los datos del LDAP de la universidad, que es único. Por cada servidor LDAP, se almacena un nombre identificativo y los datos de configuración propiamente dichos. Sólo los administradores tienen permisos para modificar esta tabla.

En la Figura 14 se muestra un esquema de las tablas y sus campos. Para ver los detalles acerca de la instalación y configuración inicial de la base de datos, puede consultar el Apéndice B.4.

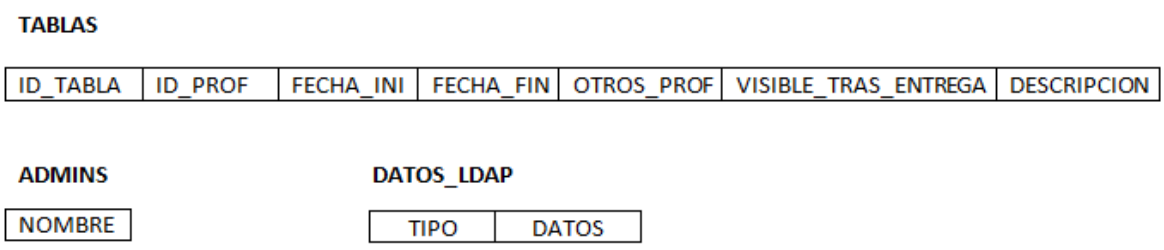


Figura 14. Tablas básicas de la base de datos

Capítulo 6

Implementación

6.1 Introducción

En este capítulo se explicarán aspectos generales de implementación, haciendo corresponder los componentes del diseño con elementos concretos de la tecnología empleada, en este caso, Java EE. Se hará una descripción más técnica de los distintos módulos de programación utilizados y se especificará su despliegue dentro del servidor Apache Tomcat, así como la funcionalidad que aportan.

Como ya se anticipó, la mayor parte de los componentes usados son JSPs (para la presentación) y servlets (para el control y procesamiento), además de alguna clase adicional para la interacción con la base de datos y el servidor de autenticación.

Se recuerda que de todos los componentes necesarios para el funcionamiento de esta plataforma (Figura 9), lo que realmente se va a implementar es la parte concerniente a la aplicación dentro del *servidor de la aplicación*, es decir, la aplicación que va dentro de Apache Tomcat. Para el resto de componentes se han utilizado módulos externos ya desarrollados que son independientes. En concreto, el cliente será un navegador Web convencional, para la base de datos se ha instalado un servidor de MySQL, y el servidor LDAP está instalado en la universidad.

6.2 Estructura de directorios

Los ficheros de la aplicación se despliegan dentro de la carpeta *webapps* de la instalación de Tomcat siguiendo las directrices que se indican en el Capítulo 3. Antes de hablar de aspectos más concretos, se ilustra un esquema global de la estructura de directorios de la aplicación (Figura 15) que ayuda a orientarse. Como se puede apreciar, en la parte pública se sitúan los ficheros concernientes a la interfaz visual: las Java Server Pages, imágenes, y la hoja de estilo CSS que da formato al código HTML.

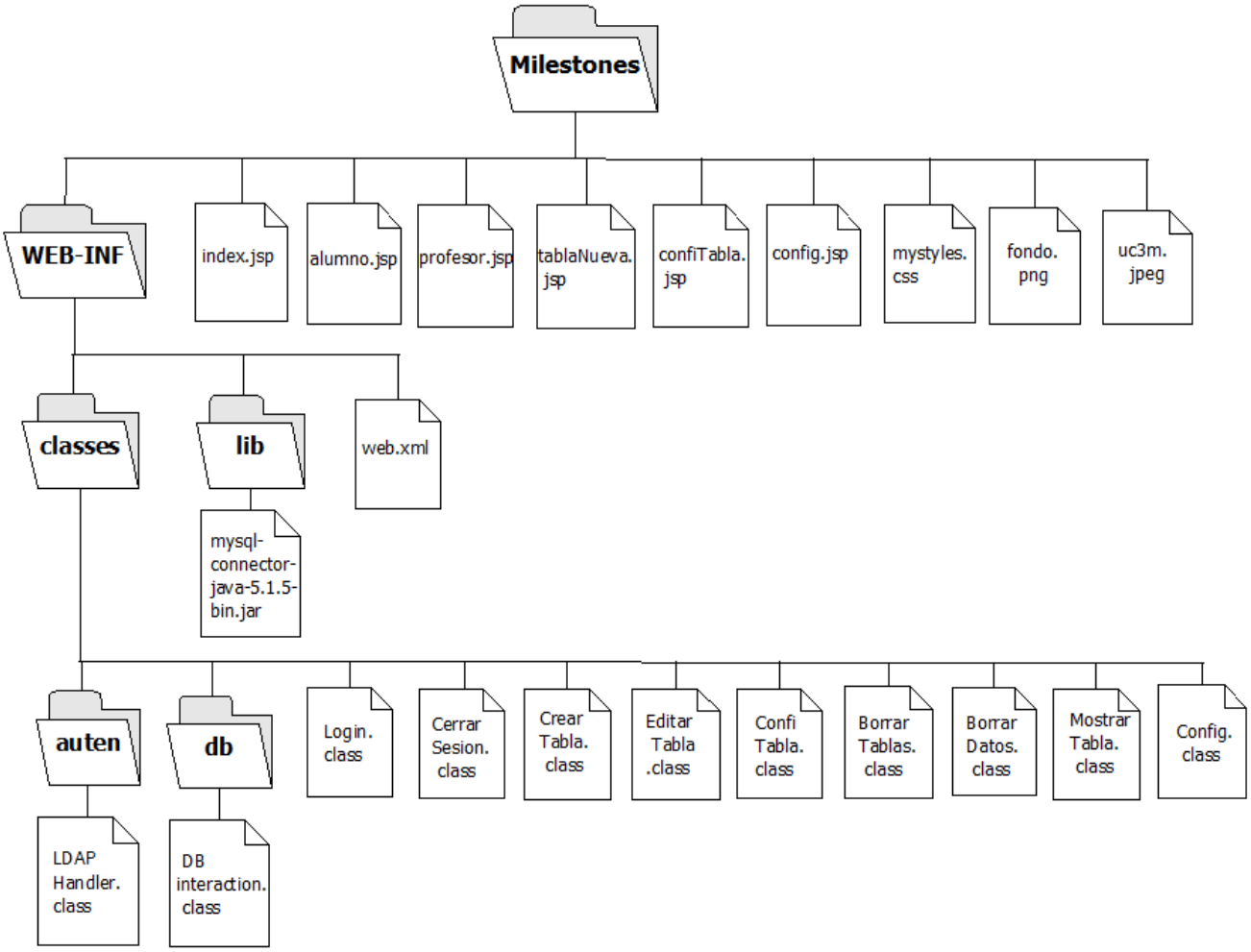


Figura 15. Despliegue de la aplicación en Tomcat

En la parte privada (WEB-INF), se sitúa el descriptor de despliegue, el driver de conectividad a la base de datos dentro de la carpeta *lib*, y la carpeta *classes*. En esta última carpeta se encuentran todos los servlets compilados que ayudan al control del flujo de la aplicación y al procesamiento de peticiones. Además, contiene a otras dos carpetas, *auten* y *db*, que contienen el manejador del acceso LDAP y el fichero de interacción con la base datos, respectivamente. Se explica todo esto más en detalle en las siguientes secciones.

6.3 Acceso Web (interfaz visual)

Como el acceso a la aplicación desarrollada se plantea mediante un navegador convencional, se ha implementado una interfaz Web para el sistema que permite la interacción con el usuario, recibiendo sus peticiones y mostrándole los resultados

oportunos en cada caso. La mayor parte de las páginas generadas tienen un esqueleto estático con alguna información que puede ser variable, como la información relativa al usuario particular o mensajes de error. En estos casos, las *Java Server Pages* resultan una solución apropiada para la generación del código HTML de manera sencilla. Para ser más estrictos a nivel técnico, todo el código generado por las JSP de nuestra aplicación son documentos XHTML ([57]) bien formados y válidos respecto a la especificación 1.1. Para comprobarlo se usó el validador de la W3C ([58],[59]).

Se ha configurado el estilo de los documentos mediante la inclusión de una hoja de estilo en cascada ([23]), el fichero *mystyles.css*. Esto independiza el contenido (la información en sí) del formato de presentación de nuestras páginas Web, facilitando el desarrollo y aportando flexibilidad. Las páginas tienen un fondo de pantalla genérico, almacenado en el archivo *fondo.png*. El logo de la universidad, que forma parte de la página inicial, está representado por el archivo *uc3m.jpeg*.

Se recuerda que las páginas JSP están en la parte pública de la aplicación, por tanto, la URL para acceder a las mismas será *http://IPservidor:8080/Milestones/pagina.jsp.* Nos centramos a continuación en cada uno de los elementos visuales que forman parte de la aplicación:

- **index.jsp**

Es la página de bienvenida de la aplicación (está configurado en el descriptor de despliegue como *wecolme file*). Esto quiere decir que se puede acceder a ella a través de la URL *http://IPservidor:8080/Milestones.* Desde esta página, tanto los profesores como alumnos podrán introducir sus datos para autenticarse y entrar en el sistema. Si los datos son válidos, se nos guía a *alumno.jsp* o *profesor.jsp*, dependiendo si eliges en el formulario el rol de alumno o profesor. Si los datos no son válidos, se muestra un mensaje de error y se vuelve a pedir las credenciales.

También se mostrará esta página si el usuario intenta acceder a cualquier otra de las páginas del sistema (p.e, para crear o ver una tabla) sin haberse autenticado previamente, o si estando dentro del sistema, hacemos *log out* para salir. En ambos casos, se muestran mensajes adicionales para requerir autenticación o informar de un correcto cierre de sesión, respectivamente. El aspecto de la página *index.jsp* puede verse en la Figura 16.

- **alumno.jsp**

La página principal para los alumnos es muy sencilla y sólo muestra una lista con aquellas encuestas en las que el alumno figura. Haciendo click en cualquier encuesta de la lista puede acceder a la URL de la misma para consultarla, actualizar sus datos o eliminar su entrada. También se incluye un enlace de *logout* para cerrar la sesión y salir del sistema.

- **profesor.jsp**

La página principal para los profesores tiene un formato similar a la de alumnos, mostrando una lista con las encuestas que gestiona el profesor que entra en el sistema. El profesor puede hacer click en una encuesta para acceder a su URL, que en este caso permite una edición avanzada de la información y de la configuración de la misma.

También puede marcar cada encuesta de la lista, y eliminar las marcadas a través de un botón. Además, se muestran enlaces para crear una nueva encuesta (nos lleva a *tablaNueva.jsp)* y para el logout. Sólo en el caso de que el profesor tenga permisos de administrador, se habilita un enlace adicional para acceder a la configuración de los servidores LDAP (fichero *config.jsp*). Véanse Figura 18 y Figura 19.

Figura 16. Página de bienvenida al sistema

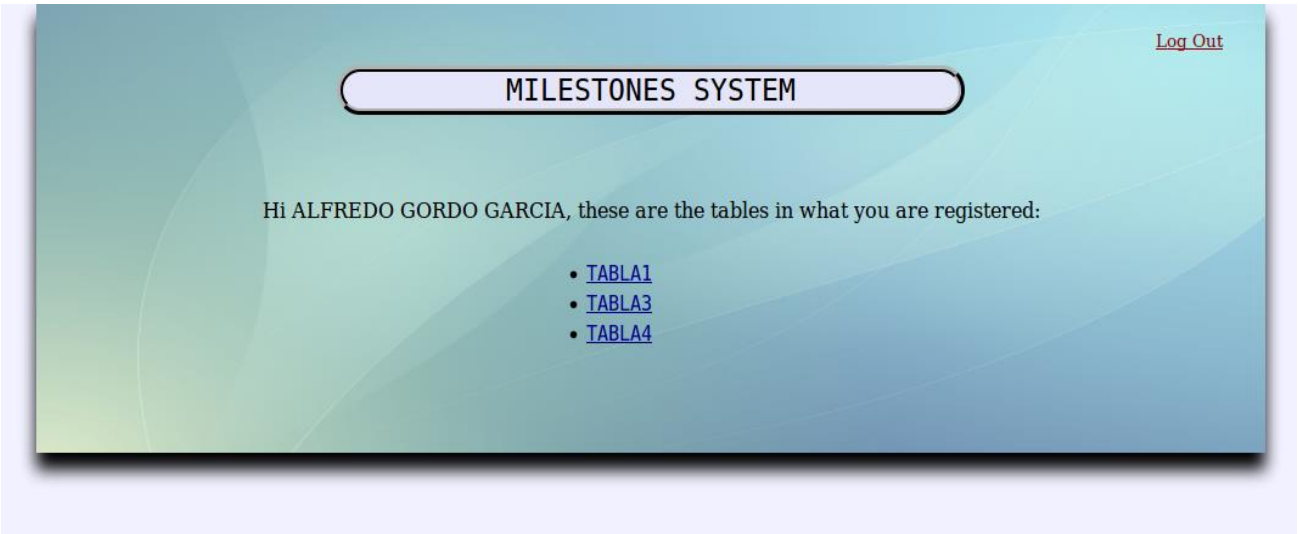


Figura 17. Página principal para alumnos

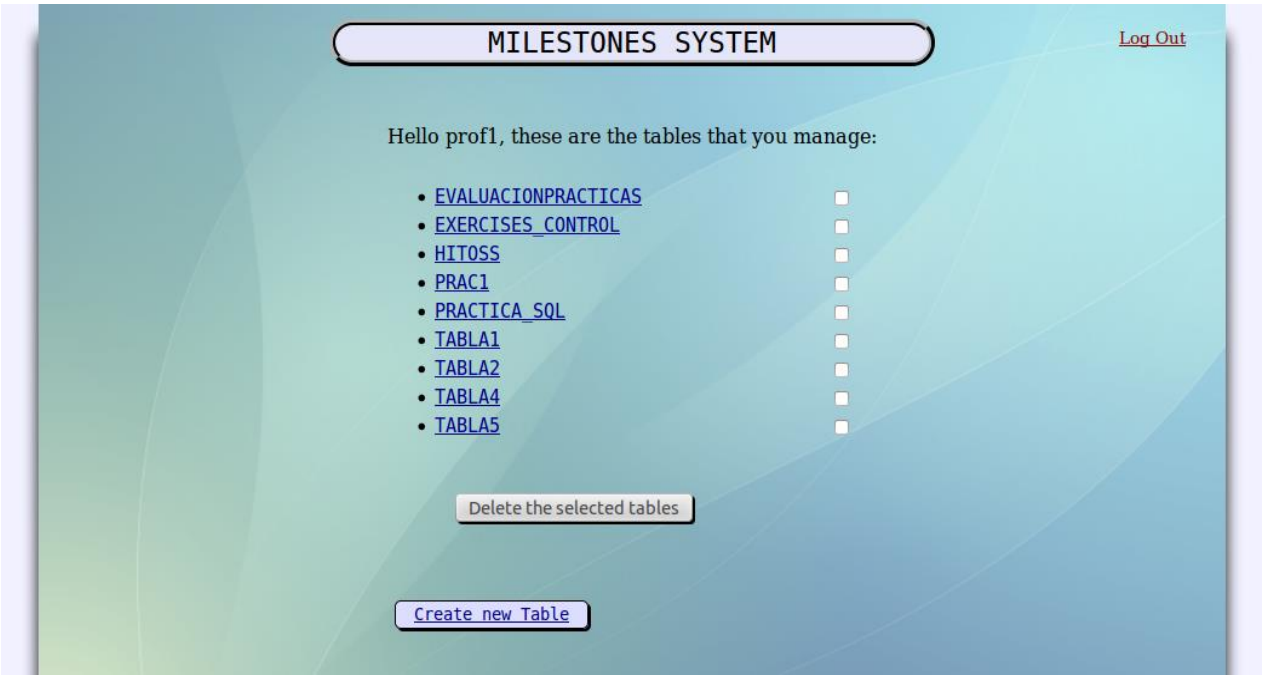


Figura 18. Página principal para profesores

Figura 19. Página principal para profesores con permiso root

- **tablaNueva.jsp**

En esta página se muestra el formulario que permite introducir la información necesaria para crear una tabla/encuesta nueva en el sistema. Se debe introducir un nombre (que debe ser único), una breve descripción textual de la encuesta, los campos o columnas de la tabla, un ejemplo de relleno para guiar a los alumnos, el período de validez con fechas de inicio y fin, una lista de *logins* de otros profesores aparte del creador a los que se permitirá la gestión de la tabla, y por último, un campo para seleccionar si se quiere que la encuesta sea visible o no para los alumnos una vez expirado el plazo de validez. Si en este campo seleccionamos "sí", el alumno podrá consultar (aunque en ningún caso modificar) la encuesta aunque esté fuera de plazo. En caso contrario, se le prohibirá el acceso a la tabla y ésta desaparecerá de su lista en la página principal del alumno. La restricción de plazos temporales es sólo para alumnos, el profesor siempre podrá consultar y modificar sus tablas aunque estén fuera de fechas. Los campos *NIA* y *GRUPO* son obligatorios para la correcta organización de las encuestas y se añaden por defecto.

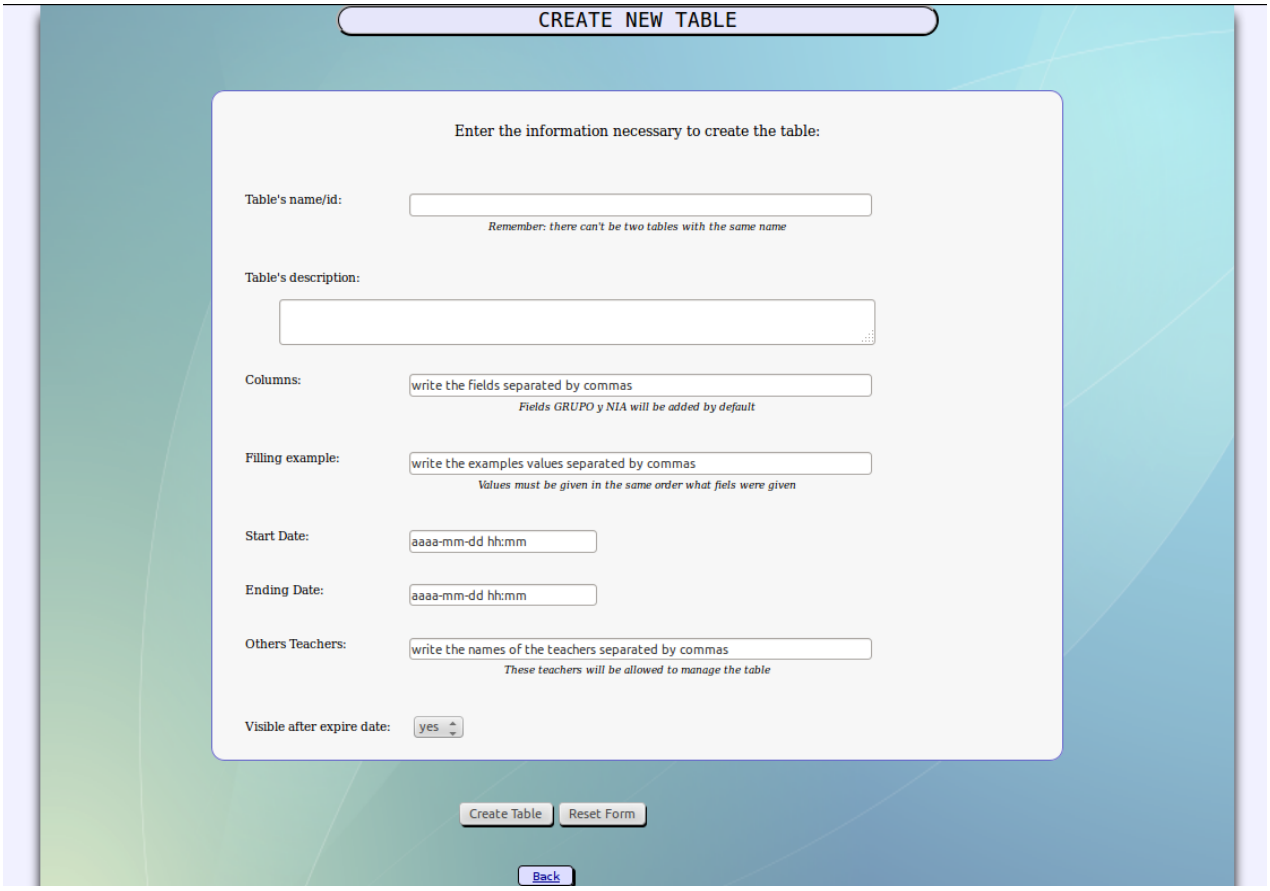


Figura 20. Página para la creación de una nueva tabla/encuesta

Al pulsar en *Create Table*, se actualizará la información en la base de datos si todo va bien, y se vuelve a la pantalla *profesor.jsp*, donde ya aparecerá la nueva tabla creada. Si la información introducida en alguno de los campos del formulario no es correcta, por

ejemplo, el nombre de tabla está repetido, el número de columnas no coincide con el número de ejemplos o el formato de fechas no es válido, se muestra la misma página con un mensaje de error. También se incluye un botón de *back* que nos lleva a *profesor.jsp*, si queremos cancelar la creación de la encuesta.

- **config.jsp**

Si el profesor es administrador, además de tener acceso a todas las tablas del sistema, puede acceder a esta página mediante el botón *settings* de su página principal. Es un formulario donde puede configurar la información de los servidores LDAP ([6]) de alumnos y profesores. Para cada uno, se puede modificar los campos fundamentales: *provider URL, Base Distinguish Name (BDN), version* y *security protocol*. Los campos aparecen ya precargados con la información existente en la base de datos.

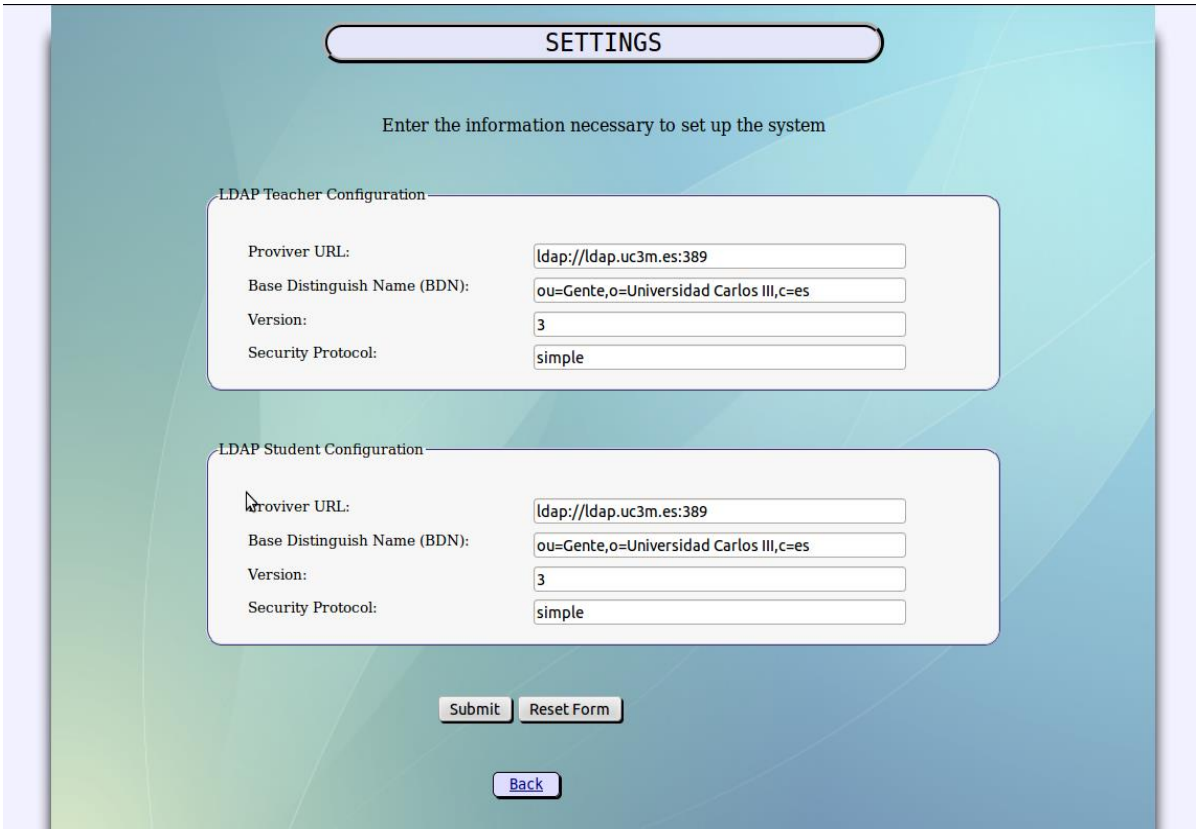


Figura 21. Página de configuración de los servidores LDAP

El campo *provider URL* es la URL del servidor LDAP, y como vemos se indica el protocolo (`ldap`), la dirección IP o el nombre de dominio (`ldap.uc3m.es`) y el puerto de red del servidor LDAP (`389`). El *BDN* indica el directorio que debemos consultar dentro de la estructura de directorios LDAP. En este caso, gente de la Universidad Carlos III situada en España. Los otros dos campos solo indican la versión y seguridad del protocolo.

- **confiTabla.jsp**

A esta página solo pueden acceder los profesores mediante un botón situado dentro de la página de vista/edición de una tabla concreta. En ella, se permite modificar algunos de los parámetros de configuración indicados en la creación de la tabla. En concreto, se permite actualizar la descripción, las fechas de validez, los profesores que tienen acceso a la tabla y el parámetro de visibilidad para los alumnos cuando la tabla está fuera de plazo. Los campos del formulario aparecen ya precargados con la información vigente en ese momento. Tras modificar los campos oportunos, pulsaremos en actualizar y veremos el cambio de los datos junto a un mensaje informativo de que la actualización se ha llevado a cabo satisfactoriamente. Si alguno de los datos introducidos no es válido, se mostrará un mensaje de error y no se hará ningún cambio. Para volver a la página de la tabla, se hará click en el botón de *back*.

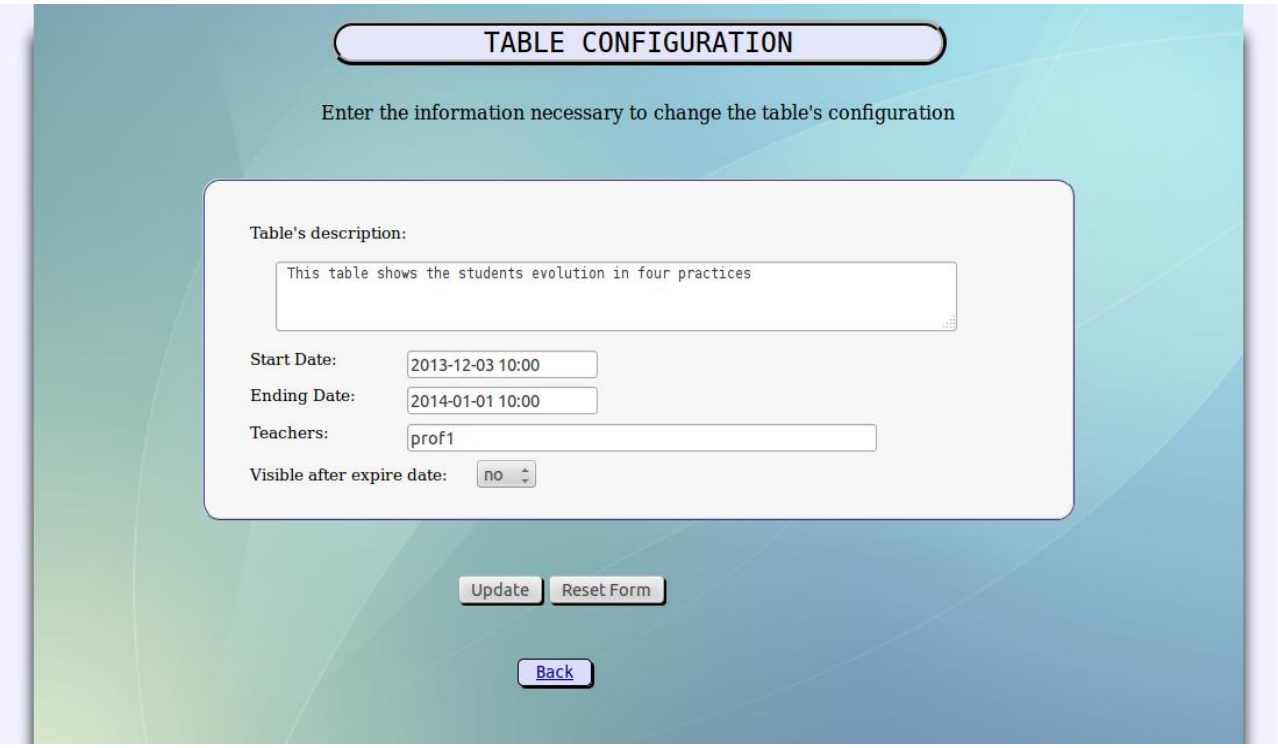


Figura 22. Página de configuración de tabla

- **Servlet MostrarTabla**

Este servlet se encarga de generar las páginas para la vista/edición de cada encuesta. Es el único servlet del sistema que realiza tareas de la interfaz visual con el usuario. La razón de haber escogido en este caso un servlet en lugar de una página JSP es que la mayoría de la información generada es variable, y además se requiere bastante procesamiento de la información. Por ejemplo, se debe cargar la información de cada tabla particular, controlar si el acceso está permitido, o permitir distintas ediciones y opciones dependiendo si se invoca como alumno o como profesor.

Las URLs de los servlets se especifican en el descriptor de despliegue *web.xml*. Una manera típica de establecerlas es mediante las URLs relativas */servlet/nombreServlet.* Para este caso en concreto, la dirección completa quedará: *http://IPservidor:8080 /Milestones/servlet/mostrarTabla*. Pero además, se debe añadir la información para saber qué tabla en concreto se quiere mostrar. Para ello, se incluye un parámetro *idTabla* cuyo valor será el identificador o nombre de la tabla que se quiere consultar. Es decir, si se quiere acceder a la tabla de nombre *TABLA1*, la URL que se debe invocar es *http://IPservidor:8080/Milestones/servlet/mostrarTabla?idTabla=TABLA1*.
Se puede acceder directamente escribiendo esta URL en la barra de direcciones del navegador, o bien mediante los enlaces de las listas de encuestas de las páginas *alumno.jsp* o *profesor.jsp*. Si escribimos la dirección directamente sin haber entrado en el sistema, se nos redirigirá a la página *index.jsp* para pedir la autenticación, y si esta es correcta, accederemos directamente a la pantalla de la tabla.

Esta pantalla es distinta dependiendo de si se entra con el rol de alumno o con el rol de profesor. Si entramos como alumno (Figura 23), se mostrarán los datos de la tabla, y al final una fila para añadir o sobrescribir información. Si el usuario ya figura en la tabla, los valores de los campos aparecerán precargados. Al pulsar *Enter or update data*, los valores se añadirán o se actualizarán en la tabla. El alumno también puede eliminar su entrada en la tabla pulsando en *Delete my entry from the table*. Se incluye un botón para volver a la página principal *alumno.jsp,* y el enlace de *logout.* Si el plazo de modificación de la encuesta ya hubiera terminado pero se permitiera la consulta para los alumnos (opción de configuración de tabla por parte del profesor), la fila y botones de edición no serán mostrados.

Figura 23. Acceso a una tabla por parte de un alumno

Si se entra como profesor (Figura 24), se permite una edición mucho más completa de la tabla en la que puede:

- Añadir o modificar entradas: en la fila de edición se hace modificable el campo *NIA*, con lo que el profesor podrá añadir más ejemplos o actualizar las entradas existentes. Para sobrescribir una entrada, solo debemos de copiar el valor de su *NIA* en el campo correspondiente de la fila de edición.

- Eliminar entradas: se puede eliminar las filas deseadas marcando el recuadro que aparece junto al NIA correspondiente, y pulsando después en *Delete the selected rows.*

- Eliminar campos: el profesor puede eliminar las columnas de la tabla que considere oportuno seleccionando los recuadros que aparecen junto a los nombres de los campos en la primera fila y luego pulsando en *Delete the selected columns.* Los campos NIA y GRUPO no se pueden eliminar pues son obligatorios.

- Añadir campos: si el profesor requiere de algún campo adicional en la encuesta puede crear una nueva columna en la tabla introduciendo su nombre en el recuadro habilitado a tal efecto y haciendo click en *Add column*. Por defecto, el valor del nuevo campo aparecerá vacío en las filas existentes.

Se incluye también un nuevo botón (*Configure Table)* que da acceso a la página *confiTabla.jsp*, en la que como ya se explicó previamente, se puede modificar algunos de los parámetros de configuración indicados en la creación de la tabla.

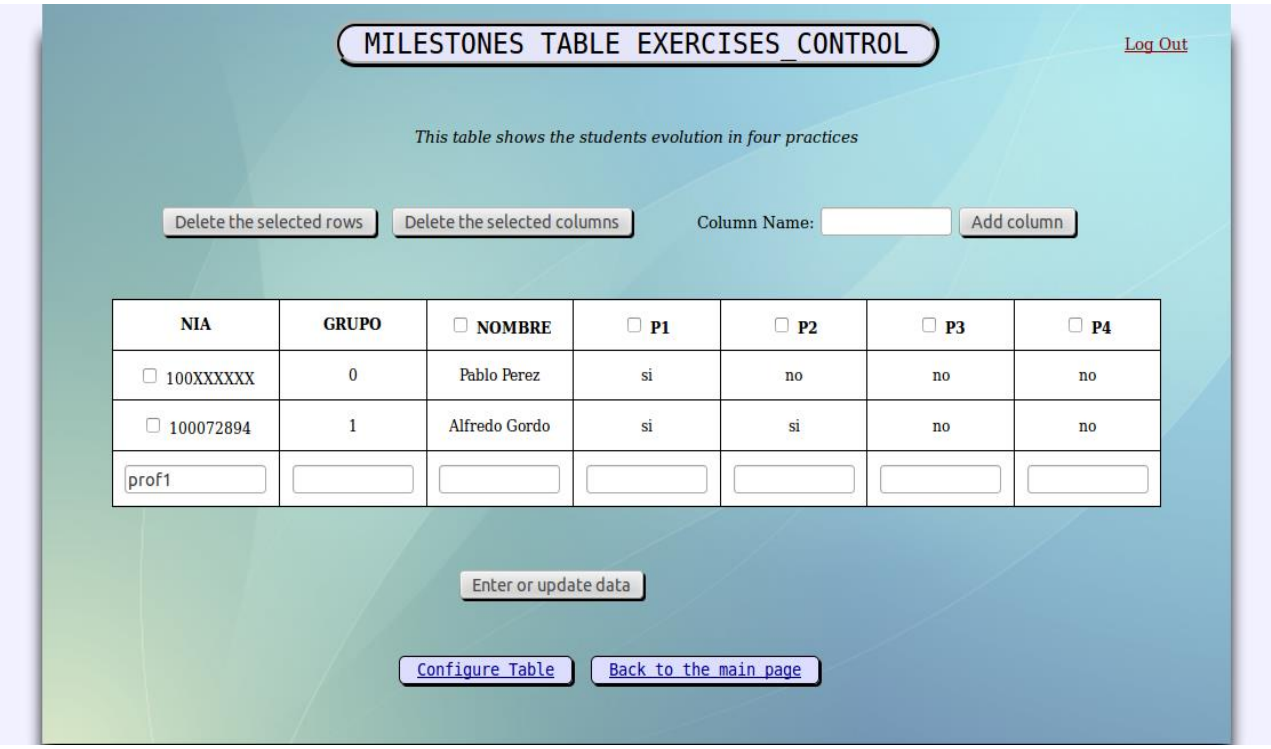


Figura 24. Acceso a una tabla por parte del profesor

En algunos casos, tanto para alumnos como profesores, se determina que el acceso a la tabla no está permitido, y entonces el servlet mostrará al usuario el siguiente mensaje:

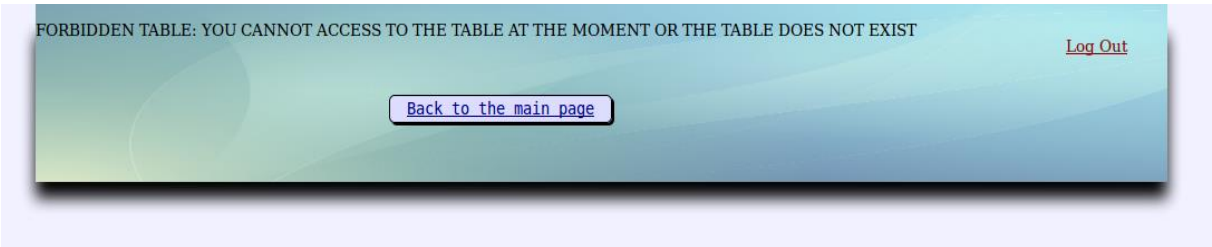


Figura 25. Intento de acceso a tabla no autorizada

Esto puede darse en varias circunstancias:

1. El identificador de la tabla pasado como parámetro al llamar al servlet no es válido o no existe.

2. Un profesor sin permisos de administrador intenta acceder a una tabla sobre la que no tiene permisos de gestión.

3. Un alumno intenta acceder a una tabla antes del comienzo de su plazo de validez.

4. Un alumno intenta acceder a una tabla una vez expirado su plazo de validez, y el profesor gestor ha configurado que no sea visible en este caso.

Hasta aquí se han explicado los elementos concernientes a la interfaz visual con el usuario. El resto de servlets del sistema forman parte del siguiente bloque.

6.4 Procesamiento y control de peticiones

Esta parte de la aplicación se compone de ocho servlets que actúan como mediadores entre las órdenes del usuario (usualmente por medio de formularios) y los componentes de *interacción con la base de datos* y *manejador LDAP*. El esquema básico de funcionamiento general de estos servlets es el siguiente:

1. Realizan un procesado parcial de las peticiones, normalmente basado en la extracción y comprobación de la validez de los parámetros

2. Si es necesaria la interacción con la base de datos o el servidor de autenticación, se delega el control a los elementos pertinentes, pasándoles la información necesaria para que se complete la operación. El manejador LDAP o la clase de interacción con la base de datos devuelven el control al servlet cuando terminan su ejecución.

3. En función de los resultados obtenidos en los puntos anteriores, controlan el flujo del programa redirigiendo hacia unas u otras páginas y/o generando mensajes de error o avisos.

Veamos cada uno de estos servlets por separado:

- **Login**

Recibe la petición del formulario de autenticación del sistema. Tras extraer los parámetros del formulario, utiliza un objeto *DBinteraction* para cargar de la base de datos la información de los parámetros del LDAP asociado al rol que se ha indicado para entrar en el sistema (alumno o profesor). Una vez hecho esto, puede crear un objeto de la clase *LDAPHandler* para el control de la autenticación. Se comprueban la validez del *login* y *password* proporcionados a través de este objeto. Si la autenticación es válida, se crea una nueva sesión, se establecen algunos atributos de sesión, y se redirige a *alumno.jsp* o *profesor.jsp* en función de rol, o bien si la autenticación venía detrás de un intento de acceso directo a una tabla concreta, se redirige a esa tabla directamente. Si la autenticación no es válida, se vuelve a *index.jsp* pasándole un mensaje de error.

- **CerrarSesion**

Este servlet es muy sencillo y solo se encarga de la tarea de destruir la sesión en el sistema. Es invocado cuando pulsamos en el enlace de *logout* desde cualquiera de las páginas de la aplicación. Invalida la sesión vigente en ese momento eliminando toda la información asociada a la misma y redirige a *index.jsp* con un aviso de cierre correcto de sesión si todo ha ido bien.

- **CrearTabla**

Se encarga del control de la creación de encuestas en el sistema. Se llama al enviar el formulario de *nuevaTabla.jsp*. Extrae los parámetros, comprueba que los obligatorios no estén vacíos, que el número de campos de la tabla a crear sea igual al número de ejemplos de relleno de los mismos, y también comprueba que el formato de fechas introducidas sea válido y que la fecha final sea posterior a la fecha inicial para evitar incoherencias. Si todo esto es correcto, se utiliza un objeto *DBInteraction* para enviarle la orden de crear la nueva tabla en la base de datos. Si todo el proceso sucede sin errores, se redirige a *profesor.jsp*. Si por el contrario ha ocurrido algún error en la comprobación de parámetros o en la propia creación de la tabla, se vuelve a mostrar el formulario de *nuevaTabla.jsp* con un mensaje de error.

- **ConfiTabla**

Este servlet se invoca al actualizar las opciones de configuración de la tabla desde el formulario de la página *confiTabla.jsp*. La filosofía es muy similar al caso anterior: comprobación de parámetros y actualización de la información en la base de datos por medio de la interfaz que proporciona la clase *DBinteraction*. Posteriormente se vuelve a redirigir a *confiTabla.jsp* con mensajes de error o actualización exitosa.

- **Config**

Servlet que es llamado al cambiar la configuración de los parámetros LDAP en el formulario de la página *config.jsp* por parte de un profesor con permisos de administrador. Extrae los parámetros del formulario comprobando que no estén vacíos y llama a la actualización pertinente en la base de datos. Después se vuelve a *config.jsp*, con mensajes de error o éxito.

- **EditarTabla**

Se encarga de gestionar las distintas acciones de edición de tabla que se pueden llevar a cabo por parte de alumnos y profesores. Lo primero que hace es comprobar qué tipo de *submit* o acción se ha invocado de entre las posibles: insertar o actualizar entrada, borrar filas, borrar columnas, o añadir columna. En función de la acción, extrae los parámetros adecuados y realiza la petición pertinente a un objeto *DBinteraction* para que realice los cambios oportunos en la base de datos. Para finalizar redirige a la URL de la misma tabla desde la que fue llamado.

- **BorrarDatos**

Este servlet es muy sencillo, sólo se ocupa se recibir la petición de un alumno cuando desea eliminar su entrada particular de una encuesta. No procesa parámetros, sólo realiza la petición a la base de datos y vuelve a la página de la tabla.

- **BorrarTablas**

En la página *profesor.jsp*, existe un pequeño formulario para marcar las encuestas de la lista que deseemos y eliminarlas del sistema. La petición de este formulario es procesada por este servlet, que extrae los nombres de las tablas a eliminar para pasárselos a un objeto *DBinteraction* que eliminará dichas tablas del sistema. Después, se vuelve a mostrar *profesor.jsp*.

6.5 Interacción con la base de datos y procesamiento de la información

Este bloque encapsula las distintas peticiones y respuestas entre la aplicación y el SGBD para realizar consultas y modificaciones en la base de datos, así como el procesado de la información de más bajo nivel, controlando ciertas condiciones y adaptando la información de la base de datos a *objetos Java* y viceversa. Incluye pues aquellos detalles concretos como los procedimientos de conexión al servidor de la base de datos o código SQL, y los agrupa en distintas funciones que pueden ser llamadas desde módulos externos, facilitando la abstracción e independencia entre los distintos componentes que forman la aplicación. Es decir, para el desarrollo de los servlets y JSPs de la interfaz visual o el bloque de procesamiento y control de peticiones, el programador no precisa conocer nada sobre la base de datos del sistema, ni siquiera sobre bases de datos en general. Sólo precisa conocer la interfaz que este módulo le proporciona.

La tecnología Java aporta la API Java DataBase Connectivity (JDBC, [56]), para tratar con bases de datos. Esta API, además, permite independizar el código del programa del sistema gestor de la base de datos, de tal manera que si en algún momento se produce una modificación del SGBD del sistema, bastará con sustituir el fichero del driver asociado al sistema gestor, sin ser necesario modificar el código del programa. En nuestro caso, se dispone del driver para MySQL, que se debe almacenar en el directorio *WEB-INF/lib* de nuestra aplicación. Es el fichero *mysql-connector-java-5.1.5-bin.jar.*

El desarrollo de este bloque se ha llevado a cabo mediante una sola clase Java, denominada *DBinteraction*, que ya se ha mencionado anteriormente. Esta clase se sitúa dentro de un paquete de nombre *db* en el directorio *WEB-INF/classes*, para una mejor estructuración de los archivos del programa. Para describir esta clase, lo mejor es explicar la interfaz que proporciona a través de sus métodos:

- *DBinteraction( )* - Constructor. Crea la conexión con el servidor de la base de datos.

- *void close( )* - Cierra la conexión con el servidor de la base de datos.

- *String getLDAPparameters(String tipoLDAP)* - Devuelve un String con los parámetros de configuración del LDAP de tipo *tipoLDAP*.

- *void configLDAP(String profs,String alums)* - Actualiza la configuración de los servidores LDAP en la base de datos. El LDAP de profesores con los parámetros guardados en *profs* y el LDAP de alumnos con los parámetros guardados en *alumns.*

- *void printStudentTables(String usr,JspWriter out)* - Consulta las tablas del alumno *usr* y utiliza el objeto *out* para imprimir por pantalla una lista de enlaces a las mismas.

- *boolean printTeacherTables(String usr, JspWriter out)* - Consulta las tablas del profesor *usr* y utiliza el objeto *out* para imprimir por pantalla una lista de enlaces a las mismas. Devuelve un valor booleano que indica si el profesor *usr* es administrador o no.

- *void crearTabla (String usr, String nombreTabla, String campos[], String valores[], String sdate, String edate, String Teachers, String visible, String descrip)* - Crea una nueva tabla/encuesta en la base de datos del sistema con los valores pasados como parámetros.

- *void borrarTablas(String tablasBorrar[])* - Elimina del sistema las tablas cuyos nombres se encuentran dentro del array de Strings *tablasBorrar.*

- *Vector mostrarTabla(String usr,String tipo, String idTabla, PrintWriter out)* - Método encargado de imprimir la tabla de nombre *idTabla* por pantalla a través del Writer *out*. Los resultados serán distintos en función del *tipo* de usuario (alumno/profesor) y el usuario concreto *usr.* Si el acceso a la tabla no está permitido para dicho usuario en función de los permisos o plazos de validez, se muestra un mensaje de error por pantalla. Además, devuelve un Vector con tres

elementos: un Vector de Strings con los nombres de las columnas de la tabla, un Boolean para indicar si el usuario *usr* tiene o no una entrada en la tabla, y un array de Strings con los valores de la entrada del usuario *usr* en el caso que la tuviera.

- *void insertarDatos(String idTabla, String[] parameters)* - Inserta en la tabla de nombre *idTabla* una nueva fila con los valores pasados en el array de Strings *parameters*. Si ya existe en la tabla un registro con el mismo valor del NIA (pasado como primer parámetro) se sobrescribe dicha fila de la tabla.

- *void borrarDatos(String usr,String idTabla)* - Elimina de la tabla con nombre *idTabla* la fila correspondiente al usuario *usr.*

- *String[] getTableData(String idTabla)* - Devuelve un array de Strings con el valor de los parámetros configurables de la tabla de nombre *idTabla* (fecha inicio, fecha fin, otros profesores, visible tras entrega y descripción).

- *void updateTable(String idTabla, String sdate, String edate, String teachers, String visible,String descrip)* - Actualiza el valor de los parámetros configurables de la tabla de nombre *idTabla* con el valor de los Strings pasados por parámetro al método.

- *void deleteRows(String idTabla,String filasBorrar[])* - Elimina de la tabla *idTabla* aquellas filas cuyo NIA se encuentre dentro del array de Strings *filasBorrar.*

- *void deleteColumns(String idTabla,String columnsBorrar[])* - Elimina de la tabla *idTabla* aquellas columnas cuyo nombre se encuentre dentro del array de Strings *columnsBorrar*.

- *void addColumn(String idTabla,String column)* - Modifica la tabla de nombre *idTabla* añadiendo una nueva columna de nombre *column.*

Para los servlets y JSPs de la aplicación, realizar operaciones sobre la base de datos es tan sencillo como crear un objeto de la clase *DBinteraction* y llamar a alguno de sus métodos con los parámetros adecuados. Todos estos métodos lanzan excepciones ante algún error en su ejecución, por lo que deben ser tratadas (bloques *try/catch*) en los servlets o JSPs correspondientes.

6.6 Manejador LDAP

La filosofía de implementación de este componente es similar al caso anterior para la interacción con la base de datos, pero ahora para interactuar con el servidor de autenticación: se pretende desarrollar una clase Java que encapsule todas las funciones concernientes al proceso de autenticación mediante LDAP ([6]), abstrayendo al resto de componentes de la aplicación de los detalles de este proceso.

En este caso, se ha reutilizado una clase Java ya desarrollada en la Universidad Carlos III de Madrid (ver Apéndice C.7), denominada *LDAPHandler*. Esta clase la ubicamos dentro de un paquete de nombre *auten* en el directorio *WEB-INF/classes* de la aplicación. Vamos a ver los métodos que ofrece:

- *LDAPHandler()* - Constructor por defecto. Llama al método *flushFields* que inicializa los atributos de configuración básicos (provider URL, BDN, version y security protocol) a los valores necesarios para acceder al servidor de autenticación LDAP de la Universidad Carlos III.

- *LDAPHandler(String purl, String bdn, String v, String sec)* - Constructor que recibe por parámetro los valores de los atributos de configuración LDAP.

- *SearchResult searchUID(String uid)* - Dado el identificador de usuario *uid* , busca en los directorios LDAP usando el filtro por defecto. Devuelve el objeto *SearchResult* asociado al *uid* si lo encuentra, o *null* en otro caso.

- *SearchResult searchUID(String uid, String filter)* - Dado el identificador de usuario *uid* , busca en los directorios LDAP usando el filtro pasado en *filter*. Devuelve el objeto *SearchResult* asociado al *uid* si lo encuentra, o *null* en otro caso.

- *boolean authenticate(SearchResult sr, String key)* - Recibe un objeto SearchResult *sr* asociado a un usuario y su contraseña (*key),* y devuelve un booleano indicando el éxito de la autenticación. Si la autenticación falla lanza una excepción.

- *boolean authenticateUID(String uid, String key)* - Realiza el proceso de autenticación recibiendo directamente el nombre de usuario (*uid)* y contraseña (*key).* Primero obtiene el *SearchResult* asociado al *uid* usando el filtro por defecto, y luego comprueba la contraseña. Devuelve un booleano indicando el éxito o fracaso de la autenticación.

- *boolean authenticateUID(String uid, String key, String filter)* - Realiza el proceso de autenticación recibiendo directamente el nombre de usuario (*uid*) y contraseña (*key*). Primero obtiene el SearchResult asociado al *uid* usando el filtro *filter,* y luego comprueba la contraseña. Devuelve un booleano indicando el éxito o fracaso de la autenticación.

Estos métodos lanzan excepciones que deberán ser tratadas en las clases donde se usen. En nuestra aplicación, la clase *LDAPHandler* se emplea en el servlet Login, encargado de gestionar el acceso al sistema. Los objetos *SearchResult* pueden utilizarse para obtener información adicional, por ejemplo, el nombre completo de los usuarios. Como se puede observar, en la clase Login, el proceso de autenticación en sí requiere sencillamente de la creación de un objeto *LDAPHandler* y la llamada al método *authenticate*, sin necesitar conocer ningún detalle de bajo nivel del funcionamiento del protocolo LDAP, consiguiendo de esta manera la abstracción buscada.

Finalizamos aquí el presente capítulo dedicado a aspectos generales de la implementación de la aplicación. Si el lector estuviera interesado en conocer detalles técnicos más concretos de la implementación, puede consultar el Apéndice C: Aspectos específicos de implementación, donde se entra más en profundidad en el código desarrollado y se explican detalladamente ciertos aspectos clave proporcionando ejemplos.

Capítulo 7

Pruebas

7.1 Funcionalidad de la aplicación

Una vez diseñada, implementada y compilada la aplicación dentro de la estructura de directorios de Apache Tomcat, se dispuso a una comprobación completa de todas sus funciones, revisando su funcionamiento y respuesta ante distintas entradas de datos por parte del usuario. También se pensó en distintas situaciones conflictivas que podrían generarse, tales como fallos de seguridad, formularios vacíos o con datos incoherentes, o acceso a tablas no autorizadas o inexistentes, y se comprobó que estaban controladas. En concreto, se realizaron los siguientes tests:

- TEST 1: no es posible ningún tipo de acceso al sistema sin autenticación previa.

- TEST 2: si un usuario se autentica como alumno, no puede acceder a ninguna funcionalidad propia de los profesores (acceso a *profesor.jsp, tablaNueva.jsp, ConfiTabla.jsp, Config.jsp*).

- TEST 3: los alumnos pueden acceder a todas las tablas a través de su URL siempre que estén en el plazo de validez. Si el plazo ya ha pasado, solo pueden ver (pero no modificar) la tabla, siempre y cuando el atributo de visibilidad tras entrega para esa tabla esté activado por parte del profesor gestor. Si dicho atributo no estuviera activado la tabla no es visible para el alumno y se muestra un mensaje de error.

- TEST 4: si un alumno intenta acceder a una tabla antes de que haya comenzado su plazo de validez, o a una tabla que no existe, se muestra un mensaje de error.

- TEST 5: un profesor solo puede acceder a las tablas que ha creado él mismo. Para poder acceder a tablas que han creado otros profesores, estos deben darle permiso en la configuración de la tabla (campo *otros profesores*). Si intenta acceder a una tabla de otro profesor que no le ha dado permiso o a una tabla inexistente, se mostrará un mensaje de error.

- TEST 6: un profesor *administrador* tiene acceso libre a todas las tablas del sistema.

- TEST 7: sólo los profesores con permisos de administración tienen acceso al formulario de configuración de los parámetros LDAP para controlar la autenticación en el sistema.

- TEST 8: en todos los formularios de los profesores se hacen comprobaciones básicas sobre la validez de los datos introducidos, por ejemplo, los campos obligatorios no están vacíos o el formato de fechas es correcto. Si los datos introducidos no son válidos y no se puede completar la acción asociada al formulario, se vuelve al mismo con un mensaje de error. Si la operación es correcta, se muestra un aviso de confirmación.

- TEST 9: ante un uso inadecuado de los controles de edición de tabla por parte de un profesor, no se produce ningún cambio en la tabla y la ejecución de la aplicación continúa normalmente.

- TEST 10: las filas de la tabla aparecen ordenadas por el número de grupo. El campo grupo debe ser siempre un número entero o no se producirá la inserción o actualización de la fila correspondiente en la tabla.

Todos los TEST se resolvieron de manera satisfactoria. Además, se comprobó la validez de la aplicación para un conjunto de doce encuestas reales, ya realizadas en papel en algún curso de la universidad, comprobando que se pueden reproducir y adaptar al sistema. Las características de estas encuestas se resumen en la Tabla 1. Por cada encuesta se indica el número de entradas (filas), número de campos (columnas), y los tipos de datos de los campos. Sólo se incluyen los tipos de datos de los campos no obligatorios (los campos NIA y GRUPO, números enteros, son comunes a todas las encuestas). Es necesario recordar que aunque aquí se indica la naturaleza de los datos de los campos de las encuestas para caracterizarlas, en la aplicación no se establece ningún control de tipos, en aras de una mayor flexibilidad y sencillez, y en la base de datos todos los campos se almacenan como cadenas de caracteres (Strings). La excepción es el campo GRUPO, que sí está controlado, pues se usa para ordenar las entradas en las tablas (TEST 10). El tipo de dato booleano es muy usado en las encuestas, y se manifiesta de varias maneras, como *ok/casilla vacía*, o *sí/no*. En ocasiones, también se admite marcar el cumplimiento de medio hito introduciendo *1/2* en estas mismas casillas. Esta es la flexibilidad que otorgan los campos tipo String que se mencionó anteriormente.

Las ocho primeras encuestas de la Tabla 1 son las más representativas, y se pueden ver en el Apéndice D: Tablas de prueba.

ENCUESTAS DE PRUEBA

Nombre Encuesta	Nº entradas	Nº campos	Tipos de datos
Control_Practica_1	12	7	string, booleano
Control_Practica_2	12	5	string, hora, booleano
Control_Ejercicios	10	7	string, booleano
Control_Ejercicios_2	26	6	string, booleano
Evaluacion_Practicas	16	4	string, número real
Practica_SQL	10	8	booleano
Control_P1	21	5	string, string+hora,booleano
Hitos	17	10	string, porcentajes
Tabla1	5	4	string,booleano
Tabla2	11	3	string,booleano
Tabla3	3	7	string, hora
Tabla4	4	6	booleano,entero

Tabla 1. Características de las encuestas de prueba

7.2 Tiempos de respuesta

Como prueba de eficiencia, se realizó la medición del tiempo de respuesta de cada una de las posibles acciones del sistema, que suelen estar asociadas a un servlet. Para ello, en Java, se usa la llamada *System.currentTimeMillis()*, que devuelve en un *long* el número de milisegundos transcurridos desde el 1 de Enero de 1970 a las 00:00 hasta el instante actual. Medir el tiempo transcurrido en una acción es tan sencillo como llamar a esta función antes y después de su ejecución, y hallar el valor de la diferencia de los valores devueltos.

Los tiempos medidos son bastante pequeños, teniendo en cuenta que tanto el servidor Apache Tomcat como la base de datos están en local, en la misma máquina que el cliente que interactúa con el sistema. También hay que tener en cuenta que los tiempos para una misma acción no son constantes, sino que dependen de los parámetros de entrada, y también de la planificación de la ejecución de procesos a bajo nivel. Por eso, para cada acción se ha hallado la media de los tiempos de cinco ejecuciones independientes, para intentar obtener un tiempo más representativo.

También, con fines comparativos, se implantó el sistema fuera de la máquina virtual en un OS Ubuntu Linux de las mismas características, y se realizaron las mismas medidas. Los resultados obtenidos figuran en la Tabla 2, Tabla 3, Tabla 4 y Tabla 5.

TIEMPOS MEDIDOS EN LA MÁQUINA VIRTUAL

Para el perfil de alumno

Acción	**Tiempo en ms**
Autenticación	1893
Mostrar Lista de Tablas	25
Mostrar Tabla	20
Insertar entrada en Tabla	96
Actualizar entrada en Tabla	135
Borrar entrada de Tabla	39
Log out	1

Tabla 2. Tiempos de respuesta para el perfil de alumno

Para el perfil de profesor

Acción	**Tiempo en ms**
Autenticación	1902
Mostrar Lista de Tablas	19
Mostrar Tabla (con edición)	39
Insertar Fila	79
Actualizar Fila	125
Eliminar Fila(s)*	78
Añadir Columna	253
Eliminar Columna(s)*	199
Cambiar configuración de Tabla	201
Borrar Tabla(s)*	90
Crear nueva Tabla	220
Modificar configuración LDAP	110
Log out	0

Tabla 3. Tiempos de respuesta para el perfil de profesor

* El tiempo fue medido para eliminar una sola fila, columna o tabla. Si eliminamos más elementos en la misma petición el tiempo se incrementa proporcionalmente.

TIEMPOS MEDIDOS FUERA DE LA MÁQUINA VIRTUAL

Para el perfil de alumno

Acción	**Tiempo en ms**
Autenticación	1503
Mostrar Lista de Tablas	21
Mostrar Tabla	22
Insertar entrada en Tabla	84
Actualizar entrada en Tabla	129
Borrar entrada de Tabla	54
Log out	0

Tabla 4. Tiempos de respuesta para el perfil de alumno (fuera de la máquina virtual)

Para el perfil de profesor

Acción	**Tiempo en ms**
Autenticación	1450
Mostrar Lista de Tablas	19
Mostrar Tabla (con edición)	35
Insertar Fila	81
Actualizar Fila	131
Eliminar Fila(s)*	101
Añadir Columna	295
Eliminar Columna(s)*	192
Cambiar configuración de Tabla	105
Borrar Tabla(s)*	110
Crear nueva Tabla	201
Modificar configuración LDAP	117
Log out	0

Tabla 5. Tiempos de respuesta para el perfil de profesor (fuera de la máquina virtual)

* El tiempo fue medido para eliminar una sola fila, columna o tabla. Si eliminamos más elementos en la misma petición el tiempo se incrementa proporcionalmente.

Como se puede apreciar, no existe una diferencia sustancial entre los tiempos de ejecución dentro y fuera de la máquina virtual. Esto es así debido a que la aplicación desarrollada es bastante ligera tanto en procesamiento como en consumo de recursos, y además la base de datos y el servidor Apache Tomcat están en local. Si se estuviera hablando de una aplicación con un uso intensivo de CPU o memoria u otros recursos, se podría notar un mayor *coste de virtualización* (pérdida de rendimiento de una aplicación al llevarla dentro de un entorno virtualizado).

Capítulo 8

Conclusiones y líneas futuras de trabajo

8.1 Conclusiones

El objetivo de este Proyecto de Fin de Carrera era el desarrollo de una interfaz Web para proporcionar un acceso electrónico al sistema de hitos enmarcado dentro del proyecto de innovación docente descrito en los primeros capítulos. Tras el diseño, implementación y pruebas de la plataforma, se puede concluir que la aplicación cumple con los requisitos especificados, y permite la adaptación Web de la estrategia de hitos, siendo viable su acoplamiento dentro de un entorno docente.

La tecnología *Java Enterprise Edition* utilizada para el desarrollo de los programas ha sido una buena opción, facilitando el desarrollo con sus muchas ventajas: gratuita, gran número de APIs, portabilidad entre plataformas y servidores, seguridad, orientación a objetos, gran comunidad de desarrolladores y disponibilidad de código externo.

En cuanto a las herramientas adicionales, *Apache Tomcat* y *MySQL Server*, se han adaptado perfectamente a la aplicación, con la ventaja de ser herramientas de código libre y disponibilidad gratuita. Ambas herramientas se instalaron y configuraron de una manera sencilla.

El desarrollo de la aplicación en un entorno virtualizado ha añadido portabilidad y flexibilidad, pudiendo desarrollar el proyecto desde cualquier PC con *VirtualBox* con tan solo transportar el *disco virtual*, y permitiendo elegir el tipo de máquina y SO en el que se quiere trabajar. La elección de un sistema Ubuntu Linux ha resultado adecuada debido a su buena integración con el resto de componentes de la aplicación. Además, se ha comprobado con las pruebas de rendimiento que como la aplicación es muy ligera, el coste de virtualización es prácticamente inexistente.

8.2 Líneas futuras de trabajo

En este proyecto se ha desarrollado la aplicación desde cero hasta el alcance de la funcionalidad deseada, que era adaptar el mecanismo del sistema de hitos en papel a una interfaz Web que permite un acceso ágil y sencillo. Las posibles líneas futuras de trabajo podrían basarse en mejorar la aplicación, desarrollando nuevas versiones que aportaran:

- **Funcionalidades adicionales**

El propio uso de la aplicación puede dar lugar a idear nuevas funcionalidades: añadir más opciones de configuración a las tablas y/o al sistema, poder agrupar encuestas por asignaturas y poder agrupar filas en las encuestas. También se puede pensar en aportar mayor flexibilidad en la creación de tablas para poder adaptar el sistema a entornos más generales (no necesariamente docentes). Podría realizarse un control más estricto de los campos con la inclusión de tipos de datos, pues en la aplicación todos los campos excepto el *grupo* son de tipo *varchar*. Además, podría añadirse la posibilidad de *tablas multidimensionales*, donde la estructura ya no debe ser necesariamente una retícula rectangular sino que pueden ser otras formas.

- **Mejora de la interfaz gráfica**

Otra línea de desarrollo podría ser mejorar la interfaz gráfica. Las páginas, formularios, y demás elementos gráficos de la aplicación pueden modificarse para ofrecer una experiencia más atractiva y un control más intuitivo al usuario, que aporte un extra de agilidad en el manejo de la aplicación. Con el desarrollo de CSS3, se abre un amplio abanico de posibilidades.

- **Hospedaje en la nube**

Debido al auge del *cloud computing* en los últimos tiempos y al avance imparable de internet, las tendencias aconsejan que hospedar tu aplicación en *la nube* puede ser el modelo del futuro. A tal efecto, ya existen plataformas como Heroku ([60]) o Amazon Web Services (AWS,[61]) que ofrecen el soporte necesario.

Apéndices

Apéndice A.

Presupuesto

1. Autor: Alfredo Gordo García

2. Departamento: Ingeniería Telemática

3. Descripción del proyecto:

- Título: Diseño e implementación de una interfaz Java EE para una aplicación multihitos
- Duración (meses): 6
- Tasa de costes indirectos: 20%

4. Presupuesto total del Proyecto: 15869 Euros

5. Desglose presupuestario (costes directos):

PERSONAL

Apellidos y Nombre	Categoría	Dedicación (hombre mes*)	Coste hombre mes (€)	Coste (€)
Basanta Val, Pablo	Ingeniero SR	0,24	4.289,54	1.029,5
Gordo García, Alfredo	Ingeniero JR	4,5	2.694,39	12.124,8
			Total:	13.154,3

* 1 Hombre mes = 131,25 horas. Máximo anual de dedicación 12 hombres mes (1.575 horas). Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas).

EQUIPO

Descripción	Coste (€)	%Uso dedicado proyecto	Dedicación (meses)	Período de depreciación (meses)	Coste imputable** (€)
Portátil con Windows7 y Office 2010	700	100	6	60	70
				Total:	70

**Fórmula de cálculo de la amortización: (A/B)·C·D donde:

A = nº de meses desde la fecha de la facturación en que el equipo es utilizado
B = período de depreciación
C = coste del equipo (sin IVA)
D = tanto por uno del uso que se dedica al proyecto

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total:	0

OTROS ASPECTOS DEL PROYECTO ***

Descripción	Empresa	Coste imputable
	Total:	0

*** Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6. Resumen de costes:

Tipo de coste	**Total (€)**
Personal	13154
Equipo	70
Subcontratas	0
Otros costes directos	0
Costes indirectos (20%)	2645
Total (€) :	**15869**

En Leganés, a __ de Junio de 2013
El ingeniero proyectista

Fdo: Alfredo Gordo García

Apéndice B.

Instalación y configuración de las herramientas utilizadas

B.1 Instalación y configuración de la máquina virtual

Una máquina virtual es un software que simula a una computadora y puede ejecutar programas como si fuese una máquina física independiente. Para desarrollar el proyecto en un sistema virtualizado se precisa de un programa que permita la creación, gestión y lanzamiento de máquinas virtuales.

VirtualBox ([8]), de Oracle Corporation, es una herramienta de virtualización de sistemas x86 y AMD64/Intel64 para uso personal y profesional. Ha sido elegido por su riqueza en características y su disponibilidad gratuita para un gran número de distintos sistemas operativos. Además, esta soportado por una activa comunidad de desarrolladores que frecuentemente añaden nuevas mejoras y lanzan nuevas versiones.

La versión utilizada en el desarrollo de este PFC fue la 4.1.12. Para instalar virtualBox en tu equipo puedes obtener el ejecutable en la sección de descargas de [8]. En el caso de Ubuntu es un paquete Debian (.deb) que puedes ejecutar directamente con el *Centro de Software de Ubuntu.* En sistemas Linux, también tienes la opción de instalarlo directamente desde los repositorios introduciendo *sudo apt-get install virtualbox* desde línea de comandos.

Una vez instalado, puedes crear y configurar tu propia máquina virtual, o bien utilizar una ya diseñada. Por ejemplo, en [62] se dispone de un variado conjunto de imágenes de máquinas virtuales basadas en kernel Linux. Para este proyecto se ha escogido esta última opción y se eligió una máquina con sistema Ubuntu Linux 12.04 x86, disponible en [63]. Una vez descargada, sólo tienes que agregarla a la lista de tus máquinas virtuales.

En todo caso, el proceso de creación de la máquina desde cero no es complicado pues el programa proporciona sencillos asistentes que te guían durante todo el proceso. En la Figura 26 se muestra la interfaz del programa, donde puede apreciarse su sencillez.

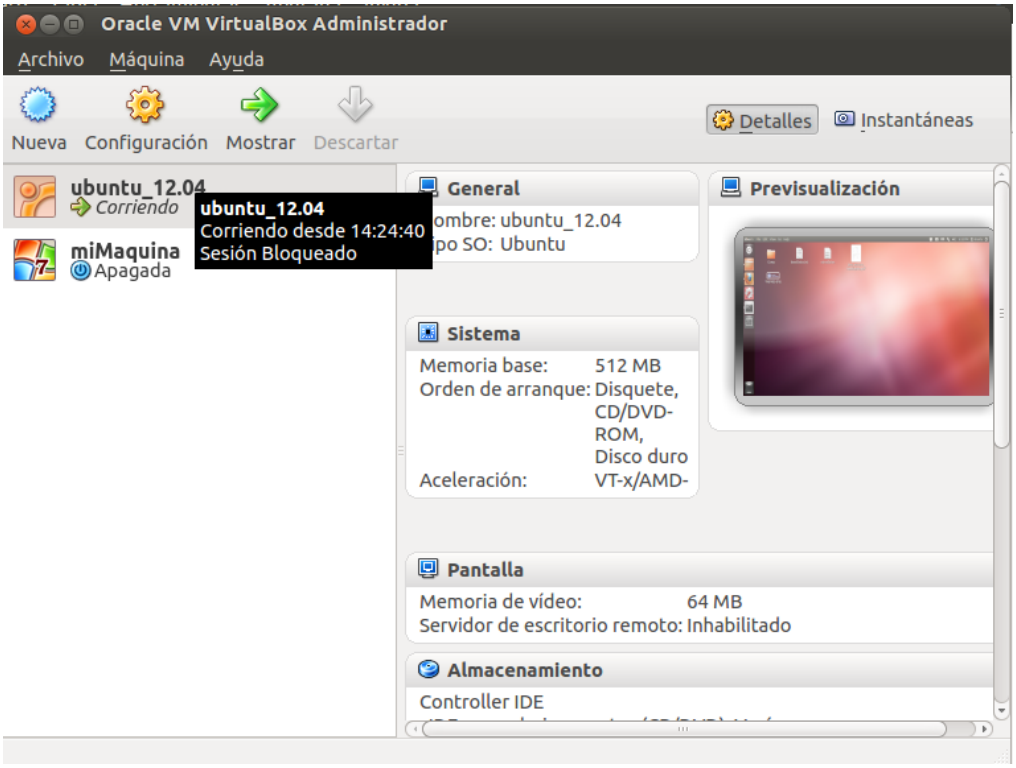


Figura 26. Interfaz gráfica de VirtualBox

Por cada máquina virtual se tiene un directorio de nombre *NombreMaquina* que contiene:

- **Carpeta logs:** aquí se van almacenado *ficheros de bitácora* con eventos e incidencias.

- **NombreMaquina.vbox:** fichero que representa a la máquina virtual y lo utiliza el programa para arrancar la máquina.

- **NombreMaquina.vdi:** es el disco duro virtual. Almacena la información y mantiene el estado de programas y ficheros del sistema virtual.

- **NombreMaquina.vbox-prev:** fichero xml de opciones de configuración del hardware de la máquina virtual, como CPU, RAM, BIOS, controladores USB, adaptador de video, y más. Normalmente no suele modificarse directamente sino a través de la interfaz del programa.

Copiando este directorio a cualquier otro ordenador se podrá ejecutar la misma máquina, siempre que el ordenador tenga instalado VirtualBox u otro software compatible.

B.2 Instalación del software de Java

Necesitamos el software de Java, en concreto el *Java Development Kit (JDK)* para desarrolladores, que incluye el entorno de ejecución *Java Runtime Environment (JRE)* y herramientas para el desarrollo, compilación, y depuración de las aplicaciones. Además se precisa el *SDK* de la parte empresarial para trabajar con los componentes Java EE.

La instalación de este software es sencilla:

1. La página oficial de Oracle proporciona la descarga conjunta del JDK 7.11 más el SDK de Java EE 6.4 en [64].

2. Descarga el archivo correspondiente a la plataforma adecuada (Linux 32 bits en este caso, llamado *java_ee_sdk-6u4-jdk7-linux.sh*).

3. Desde línea de comandos, nos situamos en el directorio donde se haya descargado el archivo y ejecutamos *bash java_ee_sdk-6u4-jdk7-linux.sh.* Esto arrancará la instalación. Puede ser necesario dar previamente permisos de ejecución (*chmod a+x java_ee_sdk-6u4-jdk7-linux.sh).*

B.3 Instalación y configuración de Apache Tomcat

En este PFC, la versión de Apache Tomcat instalada es la última versión 7.0.40. Los pasos para la instalación y configuración de Tomcat se describen a continuación:

1. Lo primero es obtener el fichero de la sección de descargas de [9], y descargarlo en el directorio que elijamos, al que nos referiremos como *${TOMCAT_HOME}.* Puede ser, por ejemplo, nuestra carpeta personal.

2. Descomprimimos el fichero.

3. Para poder usar la base de datos debemos situar el fichero *jar* del driver JDBC MySQL en el directorio *${TOMCAT_HOME}/lib* y añadirlo a la variable de entorno *$CLASSPATH*. También debemos añadir al *CLASSPATH* las APIs de Servlets y JSP que se encuentran en *${TOMCAT_HOME}/lib* (servlet-api.jar y jsp-api.jar).

4. Para arrancar el servidor, ejecutar *${TOMCAT_HOME}/bin/startup.sh* (puede ser necesario dar permisos con *chmod*) .

5. Para detenerlo ejecutar *${TOMCAT_HOME}/bin/shutdown.sh*.

Una vez arrancado, se puede comprobar su funcionamiento simplemente accediendo al mismo. Desde un navegador Web, ir a *http://localhost:8080* (ya que se está accediendo

desde el mismo ordenador). Desde cualquier otro ordenador, para acceder se debería sustituir *localhost* por la IP correspondiente al servidor. Si todo ha ido correctamente, aparecerá la pantalla de bienvenida del servidor.

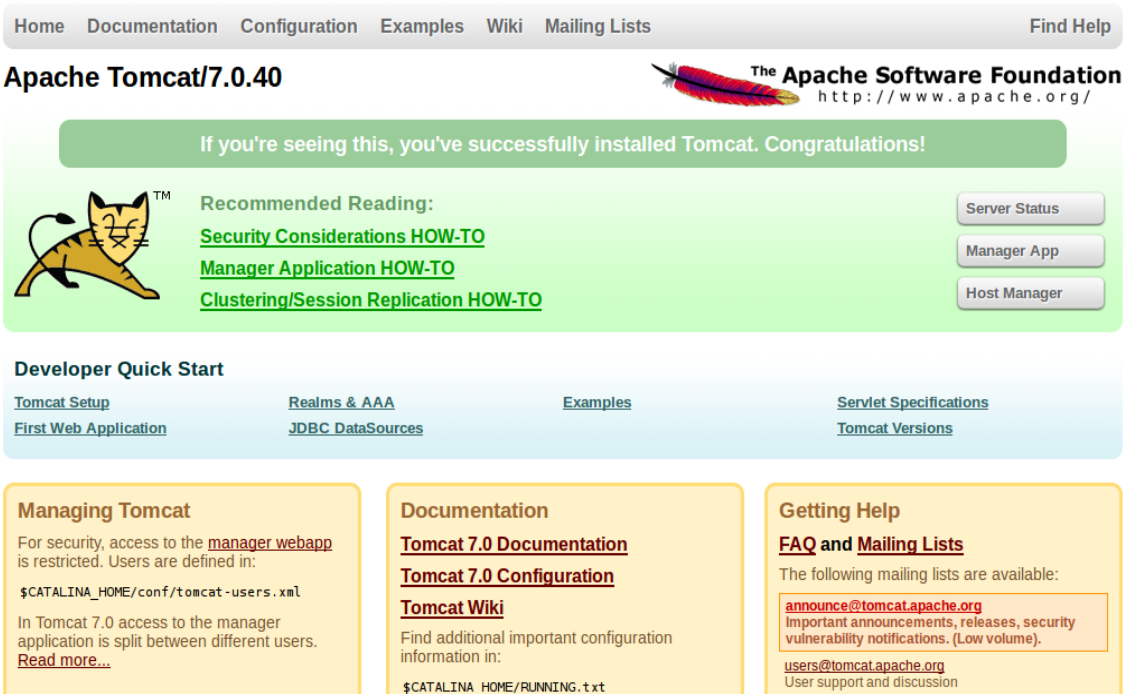


Figura 27. Pantalla de bienvenida de Apache Tomcat

Para acceder a cualquier aplicación de nombre *NombreApp* sólo tendremos que ir a *http://localhost:8080/NombreApp*. Si se produce un error al acceder al servidor, se puede comprobar su causa accediendo a los *ficheros de bitácora* (logs) de Tomcat, que se encuentran ubicados en *${TOMCAT_HOME}/logs*. Un error muy común en este paso se produce cuando el puerto 8080, puerto por defecto de Tomcat, ya se encuentra ocupado por otro proceso, y por tanto el servidor no puede escuchar en él. Si se produce esta situación, se puede configurar el puerto de escucha en el fichero *${TOMCAT_HOME}/conf/server.xml*. Simplemente busca el conector de HTTP: *<Connector port="8080" protocol="HTTP/1.1"...* y reemplaza el valor del puerto por otro disponible.

En sistemas Linux, se puede instalar Tomcat directamente de los repositorios con *sudo apt-get install tomcat7* desde línea de comandos. Se instalará como un servicio del sistema en */var/lib/tomcat7*. En este caso, se arranca con *sudo service tomcat7 start* y se para con *sudo service tomcat7 stop*.

B.4 Instalación y configuración de la base de datos MySQL

MySQL Community Server ([10]) es el SGBD escogido para el desarrollo del PFC debido a su popularidad y a que es gratuito y fácilmente configurable. Se ha utilizado en este proyecto la versión 5.5. Se puede obtener el instalable directamente de [65] para distintos sistemas operativos. Alternativamente, se puede teclear desde terminal *sudo apt-get install mysql-server* en sistemas Linux.

En el proceso de instalación se pedirá la creación de una contraseña para el administrador (usuario root), y ya tendremos configurado el servidor en nuestra máquina.

Se puede acceder a él a través de la herramienta *mysql* de línea de comandos. Si introducimos el comando *mysql -h localhost -u root –p*, se nos pedirá la contraseña (la creada en la fase de instalación), y accederemos al shell de gestión de mysql. Se puede consultar la lista de comandos disponibles con el comando *help* o *?*

- Configuración inicial del esquema relacional para la aplicación de hitos

Para dejar la base de datos lista para interactuar con nuestra aplicación se siguen los siguientes pasos:

1. **Entramos como root**

   mysql -h localhost -u root –p

2. **Creamos un nuevo esquema relacional para nuestro proyecto**:

   ```
   CREATE DATABASE APP_MULTIHITOS;
   ```

3. **Por motivos de seguridad, debemos crear un nuevo usuario** sin acceso root que tendrá privilegios sólo sobre la base de datos APP_MULTIHITOS. Las operaciones de interacción a la base de datos desde la aplicación se realizarán estableciendo la conexión con este usuario.

   ```
   /*Esto crea un nuevo usuario de nombre usuario y contraseña
   multihitos */

   CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'multihitos';

   /* Esto le otorga al nuevo usuario todos los privilegios en la
   base de datos de la aplicación. */

   GRANT ALL ON APP_MULTIHITOS.* TO 'usuario'@'localhost';
   ```

4. **Seleccionamos la base de datos creada en el paso 2**

```
USE APP_MULTIHITOS;
```

5. **Creamos las tablas básicas**

```
/* Tabla que contiene un índice con las tablas (encuestas) del
sistema e información asociada. Cada vez que creemos una tabla de
nombre NOMBRE_TABLA, deberá ser añadida aquí con ID_TABLA=
NOMBRE_TABLA */

CREATE TABLE TABLAS (ID_TABLA VARCHAR(32) NOT NULL, ID_PROF
VARCHAR(16) NOT NULL, FECHA_INI DATETIME, FECHA_FIN DATETIME,
OTROS_PROF VARCHAR(100), VISIBLE_TRAS_ENTREGA VARCHAR(2) NOT NULL,
DESCRIPCION VARCHAR(512) NOT NULL, PRIMARY KEY (ID_TABLA))
ENGINE=InnoDB;

/*Tabla donde se introducen los administradores de la aplicación*/
CREATE TABLE ADMINS (NOMBRE VARCHAR(24) NOT NULL PRIMARY KEY)
ENGINE=InnoDB;

/*Tabla para almacenar la información relativa al acceso LDAP;*/
CREATE TABLE DATOS_LDAP (TIPO VARCHAR(24) NOT NULL, DATOS
VARCHAR(100) NOT NULL, PRIMARY KEY(TIPO)) ENGINE=InnoDB;
```

6. **Introducimos datos de inicialización**

```
/* En principio solo introduciremos 2 administradores, pbasanta y
prof1 (para probar la aplicación) */

INSERT INTO ADMINS VALUES ('pbasanta');
INSERT INTO ADMINS VALUES ('prof1');

/* Introducimos los datos del servidor LDAP de la universidad */

INSERT INTO DATOS_LDAP VALUES ('profesores','ldap://ldap.uc3m.
es:389 ; ou=Gente, o = Universidad Carlos III, c=es; 3; simple');

INSERT INTO DATOS_LDAP VALUES ('alumnos', 'ldap://ldap.uc3m.
es:389; ou=Gente, o=Universidad Carlos III, c=es;3;simple');
```

Con esto, nuestra base de datos está lista para la interacción con el programa.

Apéndice C.

Aspectos específicos de implementación

C.1 Compilación de la aplicación

Nuestra aplicación se compone esencialmente de clases java (servlets, manejador LDAP e interacción con la base de datos) y JSPs. Los ficheros JSPs son transformados a servlets y compilados posteriormente a *.class* de manera automática por el motor de Tomcat. El resto de clases java debemos compilarlas manualmente. A continuación se incluyen los comandos necesarios para la compilación de todas las clases java desde la línea de comandos de un entorno Linux:

1. **Nos situamos en el directorio *classes* de nuestra aplicación.** Se supone que hemos situado aquí el código fuente del programa, aunque no es estrictamente necesario.

   cd /var/lib/tomcat7/webapps/Milestones/WEB-INF/classes/

2. **Establecemos el valor de la variable de entorno CLASSPATH**, para que el compilador *javac* sepa encontrar todos los elementos necesarios para la compilación.

   export CLASSPATH="/var/lib/tomcat7/webapps/lib/servlet-api.jar:
   /var/lib/tomcat7/webapps/lib/jsp-api.jar:
   /var/lib/tomcat7/webapps/lib/mysql-connector-java-5.1.5-bin.jar:
   /var/lib/tomcat7/webapps/Milestones/WEB-INF/classes/auten
   /LDAPHandler.java:"

3. **Escribimos la orden de compilación**

   *javac *.java*

Si hemos modificado algún fichero y recompilado, será necesario parar y volver a arrancar el servidor Tomcat para que cargue los últimos ficheros generados.

C.2 Hoja de estilos CSS

Siguiendo los consejos de desarrollo Web que dicta el *World Wide Web Consortium* (W3C, [58]) para una escritura de páginas más ordenada y correcta, se ha desarrollado código conforme a la especificación XHTML 1.1, y se separa la información de la forma de presentarla o *layout* con las hojas de estilo CSS. En nuestra aplicación se ha utilizado una sola hoja de estilo. Es sencilla, y se incluyen a continuación algunas de sus *reglas de estilo*, para ejemplificar el desarrollo de este tipo de ficheros.

```
body{

    background-color:#F1F1FF;
    background-image:url('fondo.png');
    background-repeat:no-repeat;
    background-attachment: scroll;
    background-position: center center;
    background-size: 78% 100%;
    width:78%;
    height:100%;
    box-shadow: 0px 13px 12px black;
    margin: 0 auto;
    font-size: 85%
    }


h1 {

    text-align: center;
    color: darkblue;
    width:85%;
    font-size: 26px;
    font-family:sans-serif,monospace,Verdana,Futura,Arial,Helvetica;
    font-weight: 400;
    border-style:solid;
    border-color:lightgray navy navy lightgray;
    border-width: 3px;
    border-radius:4px;
    background-color:lavender;
    margin:15px auto 30px auto;
    padding:15px 15px 15px 15px;
     }


h2{

    text-align: center;
    color: black;
    background-color:lavender;
    width:50%;
    font-size: 23px;
    font-family: monospace,serif;
    font-weight: 500;
    border: 5px ridge black;
    border-radius:19px;
    margin:15px auto 0px auto;
}
```

```
li{
    font-family:monospace;
    font-size:15px;
    text-align:left;
    position:relative;
    left:30%;
    margin:5px;
}

p.uno {
    text-align:center;
    font-size: 0.85em;
    margin:75px 0 5px 0;
}


p.dos{
    font-size: 0.85em;
    text-align: center;
    font-style: italic;
}

p.fallo {
    font-size: 0.95em;
    text-align: center;
    color:darkred

}

p.descripcion{
    font-size:1em;
    text-align: center;
    font-style: italic;

}


a{
    font-size: 1em;
    color:darkblue
}

a:hover {
     font-size: 1.05em
}


a.buttonBack{

    text-align: center;
    background-color:#ddddff;
    font-family: monospace,serif;
    border: 1px solid black;
    border-radius:5px;
    box-shadow: 2px 2px 0px 0px black;
    position:relative;
    left:40%;
    padding:2px 15px 2px 15px;
}
```

```
a.logout{

    color:darkred;
    position:absolute;
    top:4%;
    left:82%;
}

img {
    border-radius:20px;
    box-shadow: 10px 0px 12px black;
    margin:0 0 5px 0;
}


div.centrado{
    text-align: center;
}

div.centrado table {

    font-size:90%;
    border-collapse: collapse;
    margin: 50px auto 50px auto;
        background-color:white;

}

td, th {

    border: 1px;
    border-style:solid;
    border-color:black;
    padding: .8em;
  }
```

C.3 Descriptor de despliegue

El descriptor de despliegue de la aplicación Web es un fichero llamado *web.xml* que se debe incluir en el directorio *WEB-INF* de la instalación de Tomcat. Provee al contenedor información relevante de configuración de la aplicación y describe su estructura: declaración y mapeo de servlets con parámetros iniciales, configuración de sesión o gestión de alias, entre otras funciones. Se incluye a continuación el fichero concreto empleado para la aplicación multihitos:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
        Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <display-name>Milestones application teacher-student</display-name>

    <servlet>
        <servlet-name>login</servlet-name>
        <servlet-class>Login</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>login</servlet-name>
        <url-pattern>/servlet/login</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>crearTabla</servlet-name>
        <servlet-class>CrearTabla</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>crearTabla</servlet-name>
        <url-pattern>/servlet/crearTabla</url-pattern>
    </servlet-mapping>

     <servlet>
        <servlet-name>borrarTablas</servlet-name>
        <servlet-class>BorrarTablas</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>borrarTablas</servlet-name>
        <url-pattern>/servlet/borrarTablas</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>mostrarTabla</servlet-name>
        <servlet-class>MostrarTabla</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>mostrarTabla</servlet-name>
```

```
        <url-pattern>/servlet/mostrarTabla</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>editarTabla</servlet-name>
        <servlet-class>EditarTabla</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>editarTabla</servlet-name>
        <url-pattern>/servlet/editarTabla</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>borrarDatos</servlet-name>
        <servlet-class>BorrarDatos</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>borrarDatos</servlet-name>
        <url-pattern>/servlet/borrarDatos</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>cerrarSesion</servlet-name>
        <servlet-class>CerrarSesion</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>cerrarSesion</servlet-name>
        <url-pattern>/servlet/cerrarSesion</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>config</servlet-name>
        <servlet-class>Config</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>config</servlet-name>
        <url-pattern>/servlet/config</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>confiTabla</servlet-name>
        <servlet-class>ConfiTabla</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>confiTabla</servlet-name>
        <url-pattern>/servlet/confiTabla</url-pattern>
    </servlet-mapping>


    <welcome-file-list id="WelcomeFileList">
        <welcome-file>/index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

C.4 JSPs

Veamos algunos de los puntos clave en el desarrollo de este tipo de documentos:

- **Directivas iniciales de configuración**

Las siguientes directivas se incluyen al inicio de la mayoría de los ficheros JSPs creados:

```
<%@ page language="java" contentType="text/html" %>
<%@ page session="true" %>
<%@ page import="db.*"%>
```

En la primera se indica el lenguaje de programación y se establece el tipo de respuesta, en nuestro caso un documento html. La segunda es necesaria incluirla si nuestra JSP va a manejar elementos de sesión. Por último, si vamos a acceder a la base de datos, se necesita importar el paquete *db*.

- **Control de acceso no autorizado**

Como las JSPs están en la parte pública de la aplicación, se pueden dar fallos de seguridad si no se controla su acceso, por ejemplo, un usuario podría crear una tabla sin autenticarse, o un alumno podría "colarse" con perfil de profesor. Para impedir esto, por ejemplo, en la página *profesor.jsp*, se incluye el siguiente *scriptlet* a continuación de las directivas del apartado anterior. En él se comprueba que para acceder a la página debe existir una sesión creada con un atributo *login* establecido (es decir, ha habido una autenticación correcta en el sistema) y que tenemos *role* de profesor. En caso contrario se redirige a la página de bienvenida *index.jsp* con un aviso de error de autenticación. En las otras JSPs se incluyen scriptlets similares,

```
<%

String usr = "";
String nombre="";
HttpSession sesion = request.getSession();
String role=(String)sesion.getAttribute("role");

if(role!=null) {
if(role.equals("student")) sesion.setAttribute("login",null);
}

if ((sesion.getAttribute("login") == null)) {

    String aviso="<p class=\"fallo\">"+
    "Authentication Error: Please provide your login and password to log in"
    +"</p>";
    request.setAttribute("aviso",aviso);
    RequestDispatcher rd=request.getRequestDispatcher("/index.jsp");
    rd.forward(request,response);
}
```

```
else {
usr = (String)sesion.getAttribute("login");
nombre=(String)sesion.getAttribute("nombre");
}

%>
```

- **Mensajes de error o aviso**

En ocasiones, hay que mostrar notificaciones de avisos o errores en las páginas del programa. Estos mensajes se crean como Strings y se suelen almacenar como atributos de la petición o de la sesión con *setAttribute* (ver scriptlet anterior). Para recuperarlos en las JSPs e imprimirlos por pantalla se usa un scriptlet como el ejemplo siguiente:

```
<%
String aviso=(String)request.getAttribute("aviso");
if(aviso!=null)   out.println(aviso);
%>
```

- **Doctype y head**

Son iguales para todas las JSPs. Como se han seguido las directrices del W3C para generación de código XHTML, se debe incluir la referencia al DTD que incluye las reglas de validación, en este caso para XHTML 1.1. La etiqueta *html* requiere el atributo *xmlns* que especifica el espacio de nombres XML. Dentro de *head* se incluye una etiqueta *link* para indicar la ubicación de la hoja de estilos CSS que se está utilizando.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">


<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link rel="stylesheet" href="/Milestones/mystyles.css" type="text/css"/>
    <title> Milestones application Teacher-Student </title>
  </head>
```

- **Formularios**

Los formularios son elementos imprescindibles pues constituyen la manera principal de recibir información del usuario en el servidor. Como ejemplo representativo, se adjunta el código del formulario para la creación de una nueva tabla/encuesta en el sistema, y se comenta después algún aspecto sobre el mismo.

```
<form action="/Milestones/servlet/crearTabla" method="post">

<fieldset>

<h3>
Enter the information necessary to create the table:
</h3>
<br/>

<p>
<label for="nombreTabla" >Table's name/id: </label>
<input name="nombreTabla" type="text" value="" id="nombreTabla"/>
</p>

<p class="dos"> Remember: there can't be two tables with the same name </p>
<br/>

<p>
<label for="descTable" >Table's description: </label>
</p>
<textarea name="descTable" rows="6" cols="85" id="descTable"> </textarea>
<br/><br/>

<p>
<label for="campos" > Columns:</label>
<input name="campos" type="text" value="write the fields separated by commas"
id="campos" onfocus="if(this.value=='write the fields separated by commas')
this.value=''" />
</p>

<p class="dos"> Fields GRUPO y NIA will be added by default </p>
<br/>


<p>
<label for="ejem" >Filling example:</label>
<input name="ejem" type="text" value="write the examples values separated by
commas" id="ejem" onfocus="if(this.value=='write the examples values separated
by commas')this.value=''" />
</p>

<p class="dos"> Values must be given in the same order what fiels were given
</p>
<br/>


<p>
<label for="sdate" >Start Date:</label>
<input class="date" name="sdate" type="text" value="aaaa-mm-dd hh:mm"
id="sdate" onfocus="if(this.value=='aaaa-mm-dd hh:mm')this.value=''" />
</p>
<br/>


<p>
<label for="edate" >Ending Date:</label>
<input  class="date" name="edate" type="text" value="aaaa-mm-dd hh:mm"
id="edate" onfocus="if(this.value=='aaaa-mm-dd hh:mm')this.value=''" />
</p> <br/>
```

```
<p>
<label for="oTeachers">Others Teachers:</label>
<input name="oTeachers" type="text" value="write the names of the teachers
separated by commas" id="oTeachers" onfocus="if(this.value=='write the names
of the teachers separated by commas')this.value='" />
</p>

<p class="dos"> These teachers will be allowed to manage the table</p>
<br/>


<p>
<label for="visible"> Visible after expire date:</label>
<select name="visible">
  <option value="si">yes</option>
  <option value="no">no</option>
</select>
</p>

</fieldset>

<div>
<p>
<input type="submit"  value="Create Table" />
<input  type="reset"  value="Reset Form" />
</p>

</div>

</form>
```

En la etiqueta *form* se especifica la URL destino del envío en *action* y la forma de envío en *method* (GET o POST). Como vemos, la URL destino es la del servlet encargado de procesar el formulario, *Milestones/servlet/crearTabla* (se proporciona la URL relativa al servidor). Por cada campo, se crea una etiqueta *label* que asocia un trozo de texto a un control de formulario. La relación entre el control y la etiqueta se establece cuando los atributos *id* (en el control) y *for* (en la etiqueta) coinciden. En los campos tipo *input* se utiliza el atributo *onfocus*. En este atributo se escribe código utilizando un lenguaje del lado cliente, por ejemplo, JavaScript ([66]). De esta manera, cuando se dispare el evento *onfocus*, se ejecutará el código especificado en el atributo, siempre que el lenguaje utilizado sea soportado por el navegador. JavaScript es soportado por la mayoría de los navegadores actuales. El evento *onfocus* es disparado cuando un elemento recibe el enfoque, bien sea a través del ratón o por navegación tabulada. Lo que hacemos en el formulario de creación de tablas, es que cuando una entrada de texto recibe el enfoque por primera vez, eliminamos su contenido, que suelen ser instrucciones o consejos de relleno, dejándolo listo para los datos del usuario. El atributo *class*, utilizado en diversas etiquetas, se usa para dar a ese elemento un estilo específico (cuadra con un norma concreta en la hoja de estilos). Al final del formulario se incluye un control *select* (lista de opciones desplegable) y los típicos botones de *submit* y *reset*. Puede averiguar más sobre formularios HTML en [67].

- **Acceso a la base de datos**

Desde algunas JSPs es necesario acceder a la base de datos, típicamente para cargar información personal de cada usuario o precargar campos de formularios. El acceso se hace a través de un objeto *DBinteraction* y es sencillo, tal y como muestra el siguiente ejemplo, extraído de la página *profesor.jsp*:

```
<%

   DBinteraction dbi=new DBinteraction();
   boolean esRoot=dbi.printTeacherTables(usr,out);
   dbi.close();

 %>
```

Se aprecian tres pasos claros: la creación del objeto *DBinteraction*, la llamada a un método con los parámetros apropiados y por último el cierre de la conexión.

C.5 Servlets

En este apartado del apéndice, se explica más en detalle la implementación de ciertas funcionalidades de la aplicación llevadas a cabo con servlets.

- **Redirección de métodos**

Los servlets desarrollados no necesitan tener distintos comportamientos dependiendo de si la petición es de tipo GET o POST. Como habitualmente se suelen sobrescribir tanto *doGet* como *doPost*, se ha implementado en cada servlet la funcionalidad en sólo uno de los métodos, y el otro método llamará al primero, tal y como muestran los siguientes ejemplos:

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

doGet(request,response);

    }
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

      doPost(request,response);
    }
```

- **Autenticación en el sistema**

El servlet Login se encarga de la gestión del control de acceso al sistema procesando la petición del formulario inicial del sistema. Analicemos el código para entender mejor este proceso:

```
//Antes que nada, debemos importar los paquetes necesarios

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import auten.LDAPHandler;
import db.DBinteraction;
import javax.naming.directory.SearchResult;
import javax.naming.directory.Attributes;
import javax.naming.directory.Attribute;


public class Login extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

    // El método doGet llama a doPost
     doPost(request,response);
    }


  public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

    /*Lo primero que hacemos es declarar algunas variables y extraer los
    parámetros del formulario */

    String aviso;
    String nombre;
    RequestDispatcher rd;
    String tipoLDAP;
    String datosLDAP="";

    String usr=request.getParameter("login");
    String pwd=request.getParameter("pwd");
    String role=request.getParameter("role");

    // En función del rol indicado se establece el tipo de LDAP

    if (role.equals("student")) tipoLDAP="alumnos";
    else tipoLDAP="profesores";

    try{

    /* Accedemos a la base de datos para obtener los parámetros del tipo de
    LDAP establecido */

    DBinteraction dbi=new DBinteraction();
    datosLDAP=dbi.getLDAPparameters(tipoLDAP);
    dbi.close();
```

```
  //Dividimos en varios strings
   String LDAPparams[]=datosLDAP.split(";");

   //Necesitamos crear el manejador LDAP
   LDAPHandler manejador= new LDAPHandler (LDAPparams[0], LDAPparams[1],
   LDAPparams[2],LDAPparams[3]);

   /* Obtenemos el objeto searchResult asociado al usuario para obtener su
   nombre completo */
   SearchResult sr= manejador.searchUID(usr);
   Attributes atts= sr.getAttributes();
   Attribute att=atts.get("cn");

   if (att!=null) nombre= (String)att.get();
   else nombre=usr;

    //Proceso de autenticación. Se usa el método authenticate

   if(manejador.authenticate(sr,pwd)) {

   //Autenticación correcta. Creamos sesión y establecemos atributos de sesión

   HttpSession sesion = request.getSession();
   sesion.setAttribute("login",usr);
   sesion.setAttribute("nombre",nombre);
   sesion.setAttribute("role",role);

   String idTabla=(String)sesion.getAttribute("idTabla");

   /* Si se había introducido la URL de una tabla concreta, redirigiremos a
   ella directamente */

   if ( idTabla!= null) {
   rd=request.getRequestDispatcher("/servlet/mostrarTabla?idTabla="+idTabla);
   }

    else{

   //En otro caso iremos a profesor.jsp o alumno.jsp dependiendo del rol
   if(role.equals("student")) rd=request.getRequestDispatcher("/alumno.jsp");
   else rd=request.getRequestDispatcher("/profesor.jsp");

   }

   //Sentencia de redirección
   rd.forward(request,response);
 }

}


catch (Exception e) {

   /* Si salta una excepción, la autenticación es inválida. Creamos un aviso y
   redigirimos a la página principal */

   aviso="<p class=\"fallo\">"+
       "Authentication error: Invalid login or password"+
       "</p>";
```

```
    request.setAttribute("aviso",aviso);
    rd=request.getRequestDispatcher("/index.jsp");
    rd.forward(request,response);

  }

 }//doPost end
}//class end
```

Este servlet es un ejemplo muy útil ya que muestra cómo se realizan la mayoría de las acciones necesarias para nuestros servlets: extracción de parámetros de formularios, acceso a la base de datos mediante *DBinteraction*, uso del manejador LDAP mediante *LDAPHandler*, creación de sesión y establecimiento y lectura de atributos de sesión, redirecciones mediante *RequestDispatcher*, y tratamiento de excepciones y avisos.

- **Creación de nueva tabla/encuesta**

Por completitud y para fijar conceptos, ya que incluimos en el apartado anterior (JSPs) el código del formulario para la creación de una nueva tabla en el sistema, se incluye y se comenta aquí el código del servlet que recibe el envío de este formulario:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import db.DBinteraction;
//Estos dos ultimos paquetes se precisan para la comprobación de fechas
import java.util.Date;
import java.text.*;


public class CrearTabla extends HttpServlet {


    public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

        doPost(request,response);
        }


  public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

    HttpSession sesion = request.getSession(false);
    boolean datosCorrectos=true;
    String aviso="";

    if(sesion!= null){

    //Obtención de parámetros
     String usr=(String)sesion.getAttribute("login");
     String nombreTabla=request.getParameter("nombreTabla");
     String sdate=request.getParameter("sdate");
     String edate=request.getParameter("edate");
```

```
//Se realizará la comprobación del formato válido de las fechas
boolean fechasOk;

//Se establece un formato tipo que las fechas deben cumplir
SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd HH:mm");

//Se inicializan objetos Date
Date fecha_ini=new Date(0);
Date fecha_fin=new Date(0);

try{
fecha_ini=df.parse(sdate);
fecha_fin=df.parse(edate);
}
catch(Exception e){
//Si la conversión dio error, el formato introducido no es válido
fechasOk=false;
}

//Si la fecha final es anterior a la fecha inicial, no es coherente
if (fecha_fin.after(fecha_ini)) fechasOk=true;
else fechasOk=false;

String columnas=request.getParameter("campos");
String campos[]=columnas.split(",");

String ejemplos=request.getParameter("ejem");
String valores[]=ejemplos.split(",");

String Teachers= request.getParameter("oTeachers");
String visible=request.getParameter("visible");
String descrip=request.getParameter("descTable");

/* Si los parámetros obligatorios no están vacíos, el número de columnas es
igual al número de ejemplos de relleno de las mismas, y la comprobación de
fechas ha sido positiva, se procede a interactuar con la base de datos */

if (!(nombreTabla.equals("")) && !(campos[0].equals("")) &&
!(valores[0].equals("")) && (campos.length==valores.length) &&
!(sdate.equals("")) && !(edate.equals(""))&&fechasOk){

try{

/* La interacción es la típica: se crea objeto dbi, se llama al método
oportuno y posteriormente se cierra la conexión */

DBinteraction dbi=new DBinteraction();
dbi.crearTabla(usr,nombreTabla,campos,valores,sdate,edate,Teachers,visible,
descrip);
dbi.close();
    }

catch (Exception e){
/* Si ha habido error al crear la tabla en la base de datos, podemos
considerar que los datos no eran válidos */
datosCorrectos=false;

  }
} //fin del if
```

```
    else{
    datosCorrectos=false;
     }
    }

    /*Si la tabla se ha creado satisfactoriamente se redirige a profesor.jsp
    Si no, volvemos al formulario con un mensaje de error*/

    RequestDispatcher rd;

    if(datosCorrectos){
        rd=request.getRequestDispatcher("/profesor.jsp");
        rd.forward(request,response);
    }
    else{

     aviso="<p class=\"fallo\">"+
        "Please, fill the form in the right way"+
        "</p>";

    request.setAttribute("error",aviso);
    rd=request.getRequestDispatcher("/tablaNueva.jsp");
    rd.forward(request,response);

        }
    }
}
```

El resto de servlets que tratan formularios, como *Config*, *ConfiTabla* o *EditarTabla*, siguen un planteamiento muy similar.

- **Cierre de sesión**

Para llevar a cabo el cierre de sesión y salir del sistema, se ha implementado un servlet específico, *CerrarSesion*, que sólo tiene este cometido y por tanto difiere del planteamiento general de los otros servlets del sistema, que suelen encargarse de procesar formularios. Este servlet se invoca cuando, estando dentro del sistema, pulsamos en el enlace de *logout* para salir. A continuación se muestra cómo se ha implementado:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CerrarSesion extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

/*Obtenemos la referencia al objeto sesión, para llamar posteriormente al
método invalidate. Este método anula la sesión y destruye los datos asociados
a la misma */

    HttpSession sesion = request.getSession();
     sesion.invalidate();
```

```
    /*Vamos a volver a la página de bienvenida, pasándole un aviso de cierre
    correcto de sesión */

    String aviso="<p class=\"dos\">"+
    "Session sucessfully closed. Please enter your login and password to begin
    a new session"+
    "</p>";

     request.setAttribute("aviso",aviso);

      RequestDispatcher rd=request.getRequestDispatcher("/index.jsp");
      rd.forward(request,response);

    }

  public void doPost(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

         doGet(request,response);

    }

}
```

C.6 Interacción con la base de datos

En este apartado se va a hablar sobre la clase *DBinteraction*, encargada de interactuar con la base de datos. Es una clase extensa sobre la que comentaremos algunos de sus puntos más interesantes.

- **Paquetes, constantes, constructor, conexión y cierre de conexión a la BD**

```
// Lo primero es declarar el paquete al que pertenece la clase
package db;

import java.io.*;
/* Importar el paquete sql de java es imprescindible para la gestión de peticiones y respuestas a la base de datos*/
import java.sql.*;
import javax.servlet.jsp.JspWriter;
import java.util.Vector;


public class DBinteraction {

    /* Aquí se declaran tres atributos de clase (static, no dependen de la instancia concreta) cuyo valor no permite ser modificado (final), se pueden considerar constantes*/

   /* URL donde se ubica el servidor de la base de datos, especificando la base de datos concreta a la que queremos acceder*/
    private static final String URL ="jdbc:mysql://localhost/APP_MULTIHITOS";

    // Login y password para autenticación con la base de datos.
    private static final String DB_LOGIN = "usuario";
     private static final String DB_PASSWD = "multihitos";

   // Atributo que representa una conexión.
    Connection con;

     /* Constructor, se encarga de establecer una conexión con la base de datos a traves del API JDBC. Si se produce algún fallo lanzará una excepción*/
     public DBinteraction() throws SQLException, ClassNotFoundException {

        // Paso 1: Cargar el driver de jdbc
        Class.forName("com.mysql.jdbc.Driver");

        /* Paso 2: Establecer la conexión con la base de datos con
        getConnection de DriverManager. Como parámetros, recibe las constantes
        anteriormente declaradas */
        con = DriverManager.getConnection (URL, DB_LOGIN, DB_PASSWD);
    }


    /* Este método sirve para cerrar una conexión cuando ya no es necesaria, de esta manera se liberan los recursos que tenía asignados */

    public void close() throws Exception {
        con.close();
    }
```

- **Métodos**

Después del código anterior, se implementan los distintos métodos que facilitan el acceso a la base de datos a los módulos externos. El API de Java dispone de la interfaz *Statement*, que proporciona el soporte para ejecutar nuestras instrucciones SQL con dos funciones básicas:

- **`int executeUpdate(String sql) throws SQLException :`**

Ejecuta la sentencia SQL pasada por parámetro, que puede ser de tipo INSERT, DELETE o UPDATE (del *Data Manipulation Language*, DML), o bien una sentencia SQL que no devuelva nada, tales como las pertenecientes al *Data Definition Language* (DDL). El método devuelve el número de filas afectadas para las operaciones DML o bien 0 para las sentencias que no devuelven nada. Lanza una *SQLException* si ocurre algún error al acceder a la base de datos.

- **`ResultSet executeQuery(String sql) throws SQLException :`**

Ejecuta la petición SQL dada, típicamente una sentencia de consulta SELECT. Devuelve en un objeto *ResultSet* el resultado de la consulta, o bien lanza una excepción si se produce algún error.

Estas dos funciones son muy útiles para la construcción de nuestros propios métodos, que interactúan con la base de datos a través del lenguaje SQL. A continuación, a modo de ejemplo, se incluye el código comentado de algunos de los métodos desarrollados para *DBinteraction*.

```
/*Este método se encarga de borrar de la base de datos aquellas tablas cuyo
nombre esté en el array de Strings recibido como parámetro*/

public void borrarTablas(String tablasBorrar[]) throws Exception{

    Statement stmt;
    int err;

    /*El método createStatement de la clase Connection nos proporciona un
    objeto Statement*/
    stmt = con.createStatement();

    for (int i=0; i<tablasBorrar.length; i++){
        //Iteramos sobre cada uno de los elementos del array
        String idTabla=tablasBorrar[i];

        //Borramos la tabla con nombre IdTabla del indice de tablas
        err=stmt.executeUpdate("DELETE FROM TABLAS WHERE ID_TABLA ='" +
        idTabla+"'");

        //Y también la eliminamos del sistema
         err=stmt.executeUpdate("DROP TABLE "+idTabla);
        }

    //Es buena practica cerrar el statement para liberar recursos
    stmt.close();
    }
```

```
//Este método añade una nueva columna a la tabla de nombre idTabla

public void addColumn(String idTabla,String columna) throws Exception{

    Statement stmt;
    int err;
    //Se obtiene el objeto statement
    stmt = con.createStatement();

    //Se ejecuta la sentencia oportuna
    err=stmt.executeUpdate("ALTER TABLE "+idTabla+" ADD COLUMN "+columna+"
    VARCHAR(16) NOT NULL");

    //Se cierra el statement
    stmt.close();


    }
```

Muchos otros métodos que se basan en actualizar la base de datos, como *deleteColumns*, *deleteRows*, *borrarDatos*, o *updateTable*, son en estructura casi idénticos a estos dos anteriores, sólo cambia la sintaxis de la sentencia a ejecutar. Veamos ahora algún ejemplo de método donde se emplean consultas.

```
/*Este método devuelve un array de Strings con el valor de los parámetros
configurables de la tabla de nombre idTabla*/

public String[] getTableData(String idTabla) throws Exception{

    Statement stmt;
    ResultSet rs;
    String datos[]=new String[5];

    stmt=con.createStatement();

    /*Como es una consulta SELECT usamos executeQuery, que nos devuelve un
    ResultSet*/
    rs=stmt.executeQuery("SELECT FECHA_INI, FECHA_FIN, OTROS_PROF,
VISIBLE_TRAS_ENTREGA,DESCRIPCION FROM TABLAS WHERE ID_TABLA='"+idTabla+"'");

    // El método next de rs va cargando las filas que ha devuelto la consulta
    if(rs.next()){
        //Podemos referirnos a campos concretos con getString
        datos[0]=rs.getString("FECHA_INI");
        datos[1]=rs.getString("FECHA_FIN");
        datos[2]=rs.getString("OTROS_PROF");
        datos[3]=rs.getString("VISIBLE_TRAS_ENTREGA");
        datos[4]=rs.getString("DESCRIPCION");

    }

    stmt.close();
    return datos;

    }
```

```
//Método para imprimir la lista de encuestas del alumno usr

 public void printStudentTables(String usr,JspWriter out) throws Exception {

   //Declaración de variables
   Statement stmt, stmtAux;
   ResultSet rs,rsAux;
   boolean figuroEnTabla=false;

    stmt = con.createStatement();

    /* Primero debemos seleccionar las tablas permitidas (están en plazo o nos
    hemos pasado de plazo pero atributo visible= si) */

    rs=stmt.executeQuery("SELECT ID_TABLA FROM TABLAS WHERE (FECHA_INI<NOW()
    AND NOW()<FECHA_FIN) OR (NOW()>FECHA_FIN AND VISIBLE_TRAS_ENTREGA='si')");

     //Vamos a iterar sobre estas tablas
   while (rs.next()) {

        String idTabla= rs.getString("ID_TABLA");
        stmtAux=con.createStatement();

        //Por cada tabla, se consulta si tiene una entrada del alumno usr
        rsAux=stmtAux.executeQuery("SELECT * FROM "+idTabla+" WHERE NIA='"
        +usr+"'");

        if(rsAux.next()) {

               //Si rsAux.next() no es null es que hay entrada del alumno
               figuroEnTabla=true;

               //Imprimimos el item de la lista
               out.println("<li class=\"alumno\"> <a href= \"/Milestones
               /servlet/mostrarTabla?idTabla="+idTabla+"\"> "+idTabla+"
               </a></li>");

                   }
        stmtAux.close();

          }

    if(!figuroEnTabla){
    //Si el alumno no está en ninguna tabla se muestra el siguiente aviso
          out.println("<p>You are not registered in any table</p>");
          }


   stmt.close();


    }
```

C.7 Manejador LDAP

La clase que implementa el manejador LDAP, *LDAPHandler*, fue desarrollada en la Universidad Carlos III de Madrid. Se incluye aquí para el lector que tenga interés en entender los detalles de bajo nivel asociados a este proceso. Se usan librerías de Java para tablas hash (*java.util.Hashtable)* y para nombres y directorios (*javax.naming.*)*. Se recuerda también que puede consultar detalles asociados al protocolo LDAP en las referencias [6] y [7].

```
package auten;

/*
 * LDAPHandler.java
 *
 * $Id: LDAPHandler.java,v 1.2 2005/09/30 14:54:17 abel Exp $
 *
 * Copyright: This file was created at the Carlos III University of Madrid.
 * Carlos III University of Madrid makes no warranty about the suitability of
 * this software for any purpose. It is presented on an AS IS basis.
 *
 */

import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.directory.DirContext;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;

/**
 * Class to handle authentication through LDAP
 * @version $Revision: 1.2 $
 * @author Abelardo Pardo
 */

public class LDAPHandler {

  private static final String uidPlaceHolder = "@ASAPUID@";
  private static final String defaultSearchFilter="(uid="+uidPlaceHolder+ ")";

  private String provider_url;
  private String basedn;
  private String version;
  private String securityProtocol;

  public LDAPHandler() {
    flushFields();
    }

  public LDAPHandler(String purl, String bdn, String v, String sec) {

    provider_url = purl;
    basedn = bdn;
```

```
if (v != null) {
    version = v;
}
securityProtocol = sec;
}

public void flushFields() {
    provider_url = "ldap://ldap.uc3m.es:389";
    basedn="ou=Gente,o=Universidad Carlos III,c=es";
    version = "3";
    securityProtocol = "simple";
     }

/**
 * Given a uid, fetch the object data from LDAP using the default filter
 *
 * @param uid String to search for
 *
 * @return SearchResult an object with the requested uid or null if not
 * found
 *
 * @throws Exception if error during searcyh
 *
 */
public SearchResult searchUID(String uid) throws Exception {
return searchUID(uid, defaultSearchFilter);
}

/**
 * Given a uid, fetch the object data from LDAP with the given filter
 *
 * @param uid String to search for
 *
 * @param filter String to use for filtering
 *
 * @return SearchResult an object with the requested uid or null if not
 * found
 *
 * @throws Exception if error during searcyh
 *
 */
public SearchResult searchUID(String uid, String filter) throws Exception {
Hashtable env = new Hashtable();

// Create the environment
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory");

env.put(Context.PROVIDER_URL, provider_url);

// If a security protocol has been given, use it
if (securityProtocol != null) {
    env.put(Context.SECURITY_PROTOCOL, securityProtocol);
}

// If a version value has been given, use it
if (version != null) {
    env.put("java.naming.ldap.version", version);
}
```

```
    // A null or empty filter is not a valid option
    if ((filter == null) || ("".equals(filter))) {
        filter = defaultSearchFilter;
    }

    DirContext ctx = new InitialDirContext(env);
    SearchControls constraints = new SearchControls();
    constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
    NamingEnumeration results = ctx.search(basedn,
              filter.replaceFirst(uidPlaceHolder, uid),
              constraints);

    SearchResult sr = null;

    // If the obtained result object is not empty, fetch the first data
    if (results.hasMore()) {
        sr = (SearchResult) results.next();
    }

    ctx.close();
    return sr;
    }

    /**
     * Given an LDAP user object, send credentials to authenticate
     *
     * @param sr LDAP user previously obtained
     *
     * @param key password given by the user
     *
     * @return boolean Indicates the success of the authentication
     *
     * @throws Exception if authentication fails
     */

 public boolean authenticate(SearchResult sr, String key) throws Exception {

    Hashtable env = new Hashtable();

    // Create the environment
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, provider_url);

    // If a security protocol has been given, use it
    if (securityProtocol != null) {
        env.put(Context.SECURITY_PROTOCOL, securityProtocol);
    }

    // If a version value has been given, use it
    if (version != null) {
        env.put("java.naming.ldap.version", version);
    }

    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, sr.getName() + "," + basedn);
    env.put(Context.SECURITY_CREDENTIALS, key);

    DirContext ctx = new InitialDirContext(env);
```

```
    // If an exception has not happened, the authentication has
    // succeeded

    ctx.close();
    return true;
    }

    public boolean authenticateUID(String uid, String key) throws Exception {
    return authenticateUID(uid, key, defaultSearchFilter);
    }

    public boolean authenticateUID(String uid, String key, String filter)
throws Exception {

    Hashtable env = new Hashtable();
    // Create the environment
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, provider_url);

    // If a security protocol has been given, use it
    if (securityProtocol != null) {
        env.put(Context.SECURITY_PROTOCOL, securityProtocol);
    }

    // If a version value has been given, use it
    if (version != null) {
        env.put("java.naming.ldap.version", version);
    }

    // A null or empty filter is not a valid option
    if ((filter == null) || ("".equals(filter))) {
        filter = defaultSearchFilter;
    }

    DirContext ctx = new InitialDirContext(env);
    SearchControls constraints = new SearchControls();
    constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
    NamingEnumeration results = ctx.search(basedn,
                        filter.replaceFirst(uidPlaceHolder, uid),
                        constraints);
    SearchResult sr = null;

    // If the obtained result object is not empty, fetch the first data
    if (results.hasMore()) {
        sr = (SearchResult) results.next();
    }

    if (sr == null) return false;    // usr does not exist

    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, sr.getName() + "," + basedn);
    env.put(Context.SECURITY_CREDENTIALS, key);

    try {
        DirContext ctx2 = new InitialDirContext(env);
        // If an exception has not happened, the authentication has
        // succeeded
        ctx2.close();
    }
```

```
catch (Exception e) {
        // password failed
        return false;
    }

    ctx.close();
    return true;
     }
} // End of class LDAPHandler
```

Apéndice D.

Tablas de prueba

A continuación se muestran ocho tablas de prueba llevadas al sistema, adaptadas de encuestas reales ya realizadas en papel. Los NIA y nombres son inventados para proteger la identidad de los alumnos. En las tablas de la Figura 28 y Figura 29 se realiza el control del avance de dos prácticas de programación relacionadas con la implementación de aplicaciones basadas en MVC. En la Figura 30, Figura 31 y Figura 34, las tablas son similares, para el control de la realización de ejercicios. La Figura 32 muestra una tabla usada para calificar a los alumnos de un curso. En la Figura 33, se controla la realización de una práctica de base de datos y por último, en la Figura 35 se muestra el avance porcentual de cada alumno en la realización de siete tareas.

- **CONTROL_PRACTICA1**

NIA	GRUPO	NOMBRE	SERV_INSTALADO	MODELO	VISTA	CONTROL
100XXXXXX	0	Victor Jimenez	SI	SI		
100071111	1	Alvaro MAta	SI	SI	SI	
100071122	2	Victor Luengo	SI	SI	SI	
100071333	3	Virgina Benito	SI	SI		
100071444	4	Arten Ruiz	SI			
100071455	5	Jesus Regidor	SI	SI	SI	SI
100071466	6	Miguel Gordo	SI	SI		
100071467	7	Miguel Galiano	SI			
100071888	8	Lucia Fernandez	SI	SI	SI	SI
100071899	9	Irene Garcia	SI	SI	SI	
100071800	10	Lidia Almazan	SI			
100066800	11	Angel Batuecas	SI	SI	SI	SI

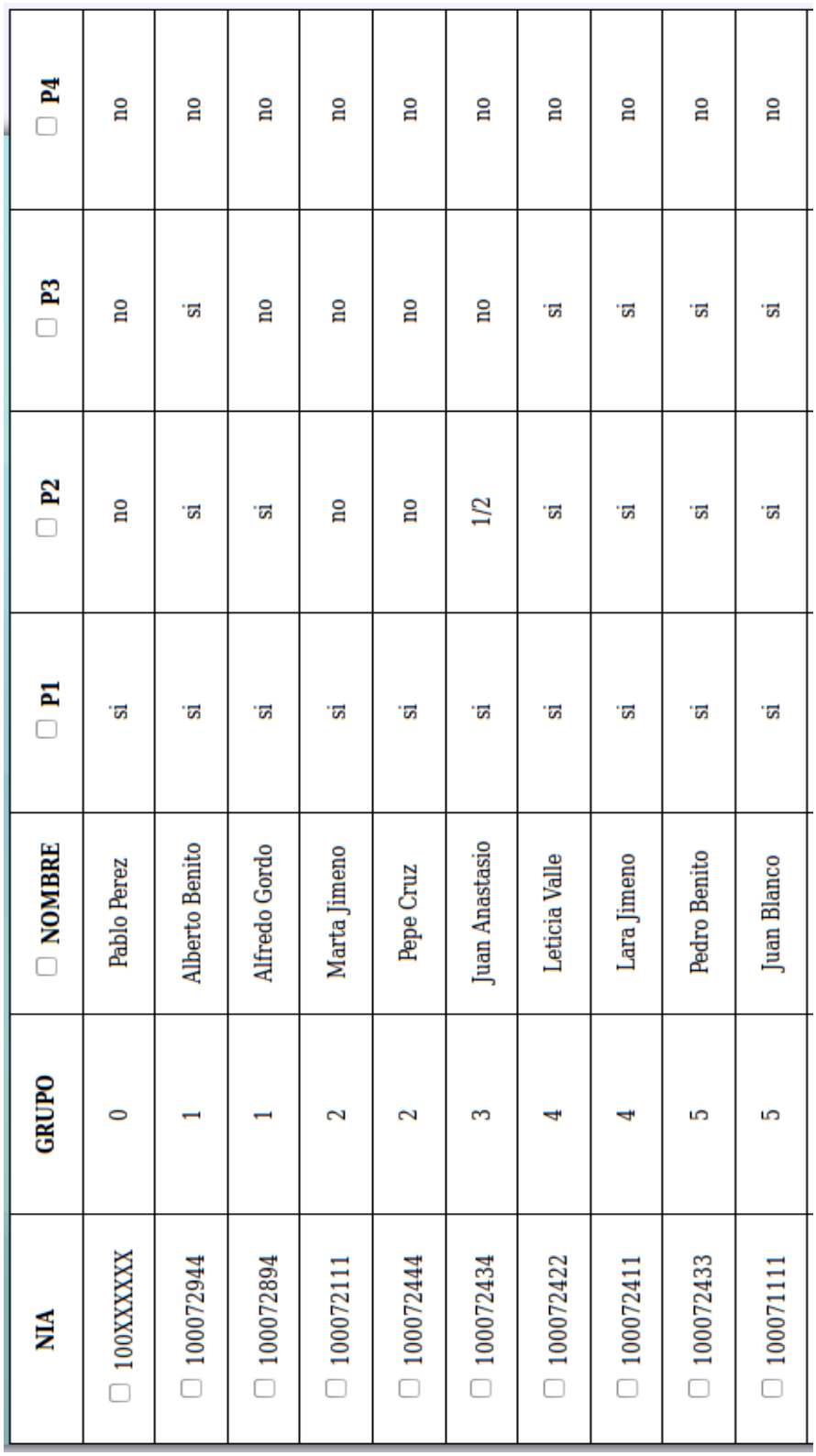
Figura 28. Tabla de Prueba Control_Practica1

- **CONTROL_PRACTICA2**

NIA	GRUPO	NOMBRE	APLICACIONFUNCIONA	MVC
100XXXXXX	0	Juan Benito	15:00	SI
10001111	1	Alberto Ruiz	14:45	NO
10002222	2	Lucia Chicharro		
100033333	3	Jose Regidor	15:15	NO
100044444	4	Jose Regidor		
100055555	5	Manuel Hernaiz		
100066666	6	Javier Morejudo		
100077777	7	Jesus Sanchez	15:40	SI
100088888	8	Victor Portillo	15:40	SI
100089999	9	Ivan Jimenez	15:40	NO
100081111	10	Daniel Rojo		
100082222	11	Daniel Rojo	15:30	SI

Figura 29. Tabla de Prueba Control_Practica2

- **CONTROL_EJERCICIOS**



NIA	GRUPO	NOMBRE	P1	P2	P3	P4
100XXXXXX	0	Pablo Perez	si	no	no	no
100072944	1	Alberto Benito	si	si	si	no
100072894	1	Alfredo Gordo	si	si	no	no
100072111	2	Marta Jimeno	si	no	no	no
100072444	2	Pepe Cruz	si	no	no	no
100072434	3	Juan Anastasio	si	1/2	no	no
100072422	4	Leticia Valle	si	si	si	no
100072411	4	Lara Jimeno	si	si	si	no
100072433	5	Pedro Benito	si	si	si	no
100071111	5	Juan Blanco	si	si	si	no

Figura 30. Tabla de Prueba Control_Ejercicios

➢ CONTROL_EJERCICIOS2

NIA	GRUPO	NOMBRE	CONFIGURACION	EJ1	EJ2
100XXXXXX	0	Marina Blanco	ok	ok	
100060000	1	Sheila Batuecas	ok		
100060001	2	Lorena Barroso	ok	ok	
100060033	3	Adrian Perez	ok	ok	ok
100060044	4	Carlos Olaya			
100060555	5	Alba Pino	ok	ok	
100062565	6	Jorge Moreno	ok	ok	ok
100062577	7	Isabel Garrido			
100062588	8	Victor Diaz	ok	ok	ok
100029999	9	Carlos Erizo	ok	ok	
100062510	10	Margarita Perez	ok	ok	
100029987	11	Mariam Paniagua	ok	ok	
100062512	12	Fernando Romero	ok	ok	
100062519	13	Fermin Azores	ok	ok	
100062514	14	Jose Corcobado	ok	ok	
100062555	15	Lucas Aldana	ok	ok	
100062556	16	Israel Marin	ok	ok	
100062500	17	Laura Garcia	ok	ok	ok
100062508	18	Luis Pardo	ok		
100062599	19	Gabriel Navarro			

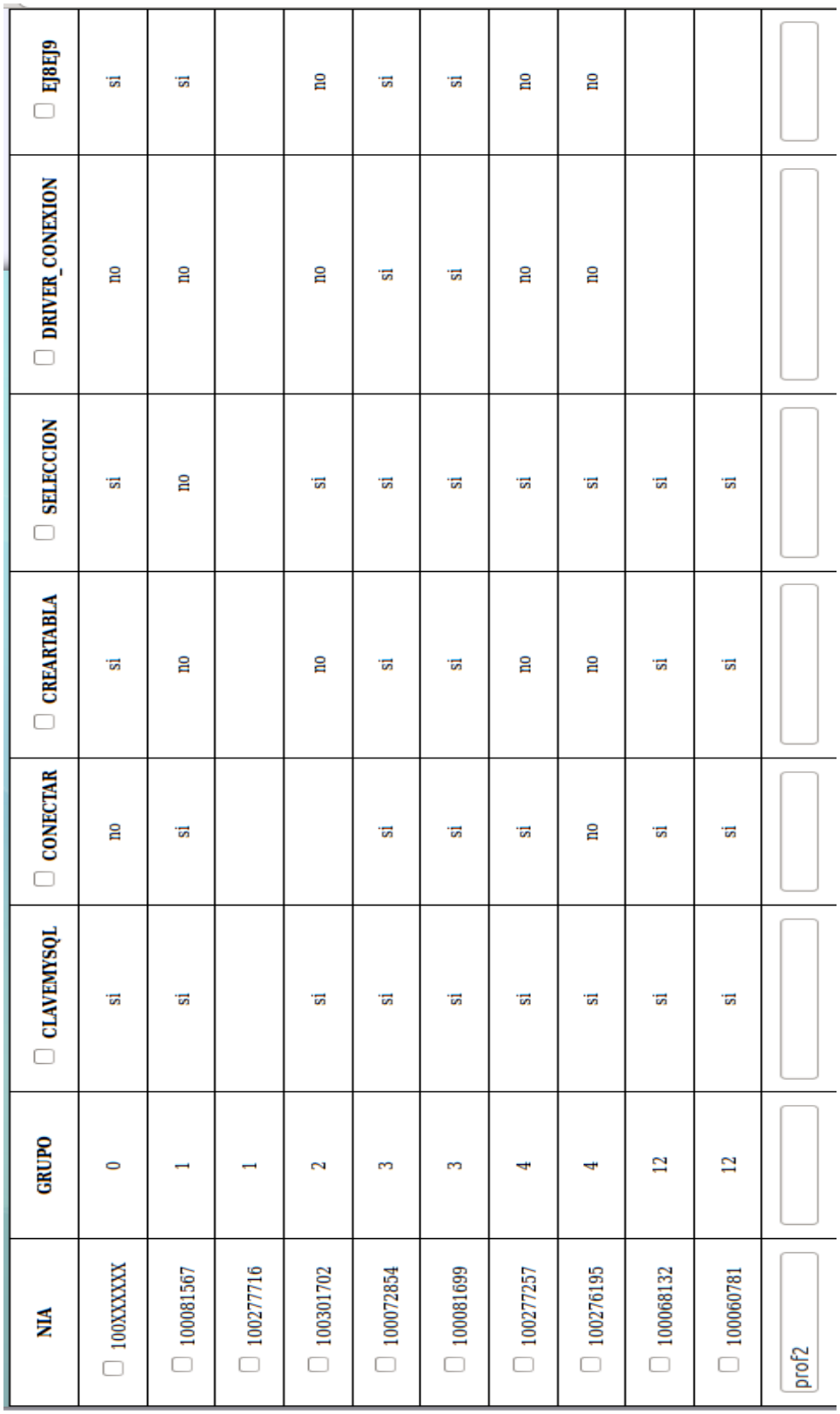
Figura 31. Tabla de Prueba Control_Ejercicios2

- **EVALUACION_PRACTICAS**

NIA	GRUPO	NOMBRE	NOTA
100XXXXXX	0	Juanjo Cruz	8
100067477	1	Miguel Antolin	8.5
100070724	1	Patricia Perez	8.4
100067645	2	Sandra Jimeno	6
100298766	3	Pedro Prieto	2.5
100072987	3	Alfredo Gomez	9
100072487	4	Maria Valverde	7
100072984	4	Alfonso Alonso	7
100071987	5	Celia Muriel	9.8
100072944	5	Alejandro Benito	9.9
100072966	6	Patrica Montero	7.8
100072555	6	Alberto Perez	7.7
100072222	7	Luis Sanchez	9.1
100072111	7	Alicia Garcia	9.1
100072511	8	Victor Gil	3.7
100072532	8	Pablo Barroso	3.7

Figura 32. Tabla de Prueba Evaluacion_Practicas

➢ PRACTICA_SQL



NIA	GRUPO	CLAVEMYSQL	CONECTAR	CREARTABLA	SELECCION	DRIVER_CONEXION	EJ8EJ9
100XXXXXX	0	si	no	si	si	no	si
100081567	1	si	si	no	no	no	si
100277716	1						
100301702	2	si		no	si	no	no
100072854	3	si	si	si	si	si	si
100081699	3	si	si	si	si	si	si
100277257	4	si	si	no	si	no	no
100276195	4	si	no	no	si	no	no
100068132	12	si	si	si	si		
100060781	12	si	si	si	si		
prof2							

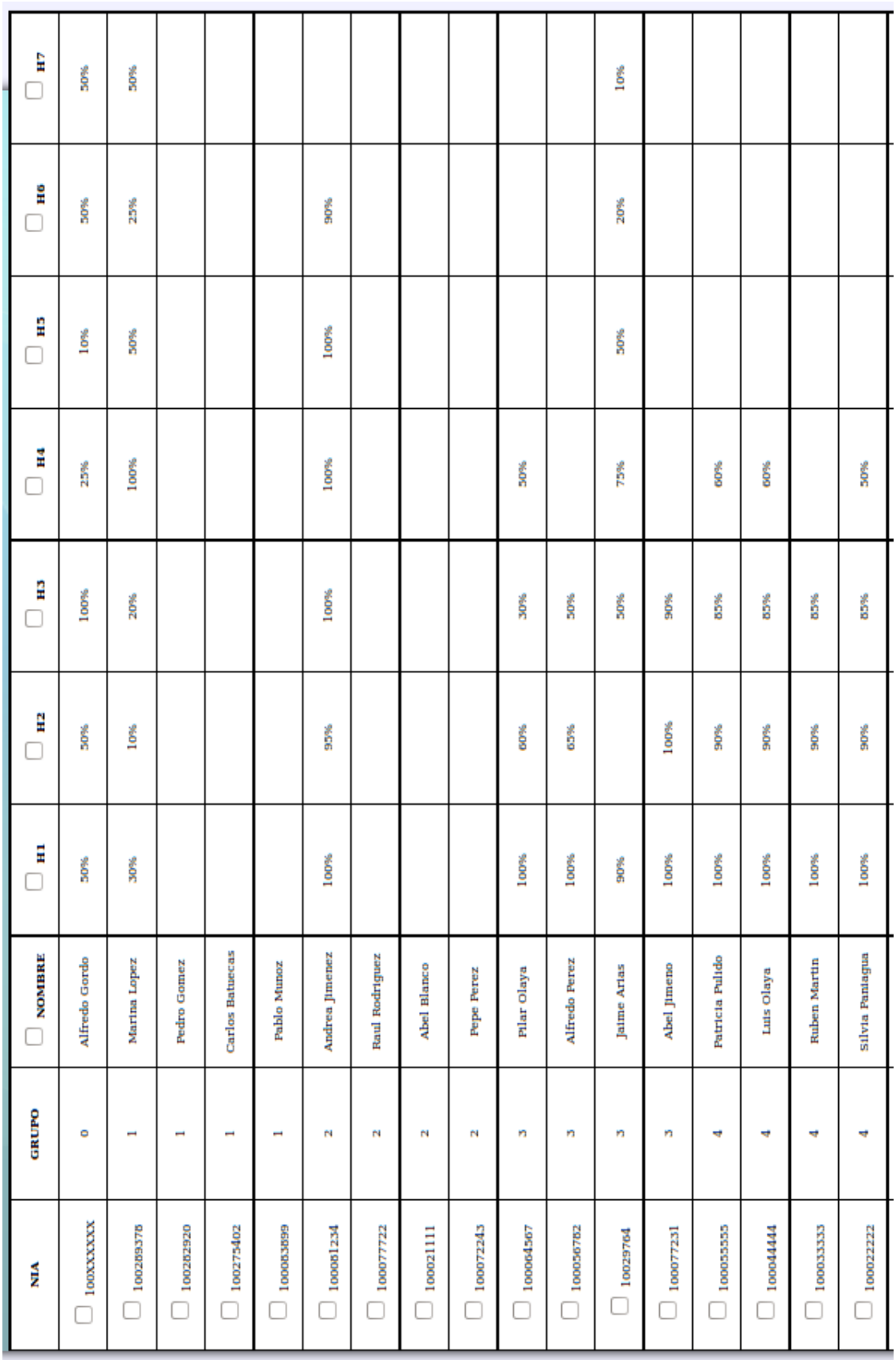
Figura 33. Tabla de Prueba Practica_SQL

- **CONTROL_P1**

NIA	GRUPO	NOMBRE	TOMCAT_INSTALADO	EJERCICIO
100XXXXXX	0	pablo basanta	instalado 14:30	ok
100079058	1	lucia chicharro		
100079152	2	Artem Suarez	instalado 15:22	
100066703	3	Josuàe regidor	instalado 15:45	ok
100079119	4	Manuel Mata	instalado 15:45	ok
100079046	5	Daniel Suarez	instalado 15:00	
100073631	6	Jesus Leon		
100066845	7	Alvaro Ruiz	instalado 15:05	ok
100072643	8	Pepe Gotera	instalado 15:30	ok
100072700	9	Alvaro Garcia		
100077777	10	Celia Perez	instalado 15:00	
100077766	11	Celia Suarez	instalado 15:05	ok
100072212	12	Paco Pizarro	instalado 15:15	ok
100077799	13	Juan Suarez		
100077788	14	Pedro Bueno	instaldo 14:45	
100077666	15	Alenjandro Lan	instalado 14:45	ok
100077199	16	Virginia Rey	instalado 15:25	
100077600	17	Gisela Caste	instalado 15:20	

Figura 34. Tabla de Prueba Control_P1

➢ HITOS



NIA	GRUPO	NOMBRE	H1	H2	H3	H4	H5	H6	H7
100XXXXXX	0	Alfredo Gordo	50%	50%	100%	25%	10%	50%	50%
100289376	1	Marina Lopez	30%	10%	20%	100%	50%	25%	50%
100282920	1	Pedro Gomez							
100275402	1	Carlos Batuecas							
100083899	1	Pablo Munoz							
100081234	2	Andrea Jimenez	100%	95%	100%	100%	100%	90%	
100077722	2	Raul Rodriguez							
100021111	2	Abel Blanco							
100072343	2	Pepe Perez							
100064567	3	Pilar Olaya	100%	60%	30%	50%			
100056782	3	Alfredo Perez	100%	65%	50%				
100029764	3	Jaime Arias	90%		50%	75%	50%	20%	10%
100077231	3	Abel Jimeno	100%	100%	90%				
100055555	4	Patricia Pulido	100%	90%	85%	60%			
100044444	4	Luis Olaya	100%	90%	85%	60%			
100033333	4	Ruben Martin	100%	90%	85%				
100022222	4	Silvia Paniagua	100%	90%	85%	50%			

Figura 35. Tabla de Prueba Hitos

Apéndice E.

Referencias

[1] *Espacio Europeo de Educación Superior (EEES).* Disponible [Internet]: http://www.eees.es/ [18 de mayo de 2013]

[2] *Tratado de Bolonia.* Disponible [Internet]: http://www.cotmec.org/bolonia.htm [18 de mayo de 2013]

[3] *Innovación Docente. Artículo.* Disponible [Internet]: http://profesores.universia.es/docencia/innovacion-docente/ [18 de mayo de 2013]

[4] Jesús Salinas. *Innovación docente y uso de las TIC en la enseñanza universitaria.* Revista Universidad y Sociedad del Consumo. Vol.1- Nº 1 Noviembre de 2004. ISSN 1698-580X Disponible [Internet]: http://www.uoc.edu/rusc/dt/esp/salinas1104.pdf [18 de mayo de 2013]

[5] *Página oficial sobre Java Enterprise Edition.* Oracle. Disponible [Internet]: http://www.oracle.com/technetwork/java/javaee/overview/index.html [19 de mayo de 2013]

[6] *Lightweight Directory Access Protocol (LDAP). Información general y enlaces a RFCs.* Disponible [Internet]: http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol [19 de mayo de 2013]

[7] *Implementación de código libre del protocolo LDAP*. OpenLDAP Foundation. Disponible [Internet]: http://www.openldap.org/ [19 de mayo de 2013]

[8] *VirtualBox*. *Página Oficial*. Oracle Corporation. Disponible [Internet]: https://www.virtualbox.org/ [19 de mayo de 2013]

[9] *Apache Tomcat.* The Apache Software Foundation. Disponible [Internet]: http://tomcat.apache.org/ [19 de mayo de 2013]

[10] *Página oficial de MySQL*. Oracle Corporation. Disponible [Internet]: http://www.mysql.com/ [19 de mayo de 2013]

[11] *Máquina Virtual*. *Definición y tipos.* Disponible [Internet]: http://es.wikipedia.org/wiki/M%C3%A1quina_virtual [19 de mayo de 2013]

[12] Isabel Martín. *Ventajas y desventajas de la virtualización*. Artículo del 23 de Mayo de 2008. Disponible [Internet]: http://www.techweek.es/virtualizacion/tech-labs/1003109005901/ventajas-desventajas-virtualizacion.1.html [19 de mayo de 2013]

[13] *Linux.* Web Oficial. Disponible [Internet]: http://www.linux.org/ [19 de mayo de 2013]

[14] *Ubuntu.* Web Oficial. Disponible [Internet]: http://www.ubuntu.com/ [19 de mayo de 2013]

[15] *Servidor Web.* Descripción. Disponible [Internet]: http://es.wikipedia.org/wiki/Servidor_web [19 de mayo de 2013]

[16] *The HyperText Transfer Protocol (HTTP)*. Descripción breve. Disponible [Internet]: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol [19 de mayo de 2013]

[17] RFC 2616: *Hypertext Transfer Protocol -- HTTP/1.1*. Disponible [Internet]: http://tools.ietf.org/html/rfc2616 [19 de mayo de 2013]

[18] *Base de Datos Relacional*. Artículo. Disponible [Internet]: https://en.wikipedia.org/wiki/Relational_database [19 de mayo de 2013]

[19] *¿Qué es un sistema gestor de base de datos o SGBD?* Disponible [Internet]: http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/ [19 de mayo de 2013]

[20] Dr. H. Störrle, Dr.A. Knnap. *Unified Modeling Language (UML) 2.* Tutorial. University of Munich. 2005. Disponible [Internet]: http://www.pst.ifi.lmu.de/veroeffentlichungen/UML%202.0-Tutorial.pdf [20 de mayo de 2013]

[21] *Software Development Kit (SDK)*. Definición. Disponible [Internet]: http://en.wikipedia.org/wiki/Software_development_kit [20 de mayo de 2013]

[22] *XML*. Tutorial. w3schools. Disponible [Internet]: http://www.w3schools.com/xml/xml_whatis.asp [20 de mayo de 2013]

[23] *Hojas de estilo en cascada (CSS)*. *Home Page.* W3C. Disponible [Internet]: http://www.w3.org/Style/CSS/ [20 de mayo de 2013]

[24] *Dirección IP*. ¿Qué es? Disponible [Internet]: http://es.kioskea.net/contents/267-direccion-ip [20 de mayo de 2013]

[25] *Puertos de aplicación*. ¿Qué son? Disponible [Internet]: http://es.kioskea.net/contents/272-puerto-puertos-tcp-ip [20 de mayo de 2013]

[26] *Oracle Corporation*. Web oficial. Disponible [Internet]: http://www.oracle.com/index.html [21de mayo de 2013]

[27] *Arquitectura de n niveles*. Software y aplicaciones Web. Disponible [Internet]: http://www.jtentor.com.ar/post/Arquitectura-de-N-Capas-y-N-Niveles.aspx [21de mayo de 2013]

[28] *El lenguaje de programación Java*. Tutorial. Disponible [Internet]: http://www.abaco.edu.pe/Manuales%5CLenguaje%20de%20programaci%C3%B3n%20Java%5CEl%20lenguaje%20de%20programaci%C3%B3n%20Java%20NXT.pdf [21de mayo de 2013]

[29] S.Pickin, N.Martinez, P.Basanta. *Introducción a las arquitecturas de componentes y a Java EE*. Curso de Software de comunicaciones. Universidad Carlos III de Madrid. Disponible [Internet]: http://ocw.uc3m.es/ingenieria-telematica/software-de comunicaciones/transparencias/3_cmpnts-JavaEE.pdf/view [21de mayo de 2013]

[30] *The Java Community Process (JCP)*. Web oficial. Disponible [Internet]: http://www.jcp.org/en/home/index [22 de mayo de 2013]

[31] *Java Standard Edition (JSE)*. Página oficial. Oracle. Disponible [Internet]: http://www.oracle.com/technetwork/java/javase/overview/index.html [22 de mayo de 2013]

[32] *Application Programming Interface (API)*. Definición. Disponible [Internet]: http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones [22 de mayo de 2013]

[33] *Java EE API Specification*. Documentación de Oracle. Disponible [Internet]: http://docs.oracle.com/javaee/6/api/overview-summary.html [22 de mayo de 2013]

[34] *Java Remote Method Invocation (RMI)*. Documentación de Oracle. Disponible [Internet]: http://docs.oracle.com/javase/tutorial/rmi/index.html [22 de mayo de 2013]

[35] *Java Server Faces (JSFs)*. Oracle. Disponible [Internet]: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html [22 de mayo de 2013]

[36] *Servicios Web (Web Services)*. W3schools. Tutorial. Disponible [Internet]: http://www.w3schools.com/webservices/ws_intro.asp [22 de mayo de 2013]

[37] *El mapeo objeto-relacional (ORM)*. Artículo. Disponible [Internet]: http://objectfreezer-r.sourceforge.net/2%20Mapeo%20objeto-relacional.html [22 de mayo de 2013]

[38] *Sistemas de transacciones distribuidas*. Documento. Disponible [Internet]: http://cs.uns.edu.ar/~sd/data/apuntes/SD-2011-mod%2009.pdf [22 de mayo de 2013]

[39] *Enterprise JavaBeans (EJBs).* Artículo. Disponible [Internet]: http://es.wikipedia.org/wiki/Enterprise_JavaBeans [23 de mayo de 2013]

[40] *Common Object Request Broker Architecture (CORBA).*Web oficial. Disponible [Internet]: http://www.corba.org/ [23 de mayo de 2013]

[41] *Simple Object Access Protocol (SOAP).* Tutorial. W3schools. Disponible [Internet]: http://www.w3schools.com/soap/soap_intro.asp [24 de mayo de 2013]

[42] S.Pickin, N.Martinez, F. Almenárez, P.Basanta. *Nivel de presentación. Aplicaciones Web.* Curso de Software de comunicaciones. Universidad Carlos III de Madrid. Disponible [Internet]: http://ocw.uc3m.es/ingenieria-telematica/software-de-comunicaciones/transparencias/5_nivelpresentacion.pdf/view [24de mayo de 2013]

[43] *Common Gateway Interface.* Artículo. Disponible [Internet]: http://en.wikipedia.org/wiki/Common_Gateway_Interface [24 de mayo de 2013]

[44] *API Specification javax.servlet*. Documentación de Oracle. Disponible [Internet]: http://docs.oracle.com/javaee/6/api/javax/servlet/package-summary.html [24 de mayo de 2013]

[45] *API Specification javax.servlet.http* Documentación de Oracle. Disponible [Internet]: http://docs.oracle.com/javaee/6/api/javax/servlet/http/package-summary.html [24 de mayo de 2013]

[46] *Applets.* Tutorial. Oracle Corporation. Disponible [Internet]: http://docs.oracle.com/javase/tutorial/deployment/applet/ [24 de mayo de 2013]

[47] *Multipurpose Internet Mail Extensions (MIME).* Tipos. Disponible [Internet]: http://www.htmlquick.com/es/reference/mime-types.html [25 de mayo de 2013]

[48] *Java Server Pages.* Tutorial. Disponible [Internet]: http://www.jsptut.com/ [25 de mayo de 2013]

[49] *Java Server Pages*. Sintaxis normal y XML. Disponible [Internet]: http://www.tutorialspoint.com/jsp/jsp_syntax.htm [25 de mayo de 2013]

[50] *Patrón Modelo-Vista-Controlador*. Artículo. Disponible [Internet]: http://prestashop5estrellas.wordpress.com/2010/03/29/el-patron-mvc-modelo-vista-controlador/ [26 de mayo de 2013]

[51] *Presentación de la tecnología Java EE*. Universidad de Huelva. Disponible [Internet]: http://www.uhu.es/josel_alvarez/NvasTecnProg/recursos/tTema1.pdf [26 de mayo de 2013]

[52] *Página de descarga de un curso sobre Java EE.* Disponible [Internet]: http://www.identi.li/index.php?topic=193837 [26 de mayo de 2013]

[53] *The Apache Software Foundation (ASF)* . Disponible [Internet]: http://www.apache.org [26 de mayo de 2013]

[54] *James Duncan Davidson.* Biografía. Disponible [Internet]: http://en.wikipedia.org/wiki/James_Duncan_Davidson [29 de mayo de 2013]

[55] *Documentación sobre el descriptor de despliegue web.xml*. Disponible [Internet]: http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html [30 de mayo de 2013]

[56] *Java DataBase Connectivity (JDBC)*. Documentación.Oracle. Disponible [Internet]: http://www.oracle.com/technetwork/java/overview-141217.html [3 de Junio de 2013]

[57] *eXtensible HyperText Markup Language (XHTML)*. Disponible [Internet]: http://www.w3.org/MarkUp [8 de Junio de 2013]

[58] *World Wide Web Consortium (W3C)*. Web oficial. Disponible [Internet]: http://www.w3.org/ [8 de Junio de 2013]

[59] *Validador XHTML de la W3C*. Disponible [Internet]: http://validator.w3.org/ [8 de Junio de 2013]

[60] *Plataforma Heroku para la implantación de aplicaciones en la Web*. Disponible [Internet]: https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat [20 de Junio de 2013]

[61] *Amazon Web Services(AWS)*. Servicios de hospedaje en la nube. Amazon. Disponible [Internet]: http://aws.amazon.com/es/free/ [20 de Junio de 2013]

[62] Imágenes de máquinas virtuales basadas en kernel Linux. VirtualBoxes. Disponible [Internet]: http://virtualboxes.org/images/ [20 de Junio de 2013]

[63] Imagen de la máquina virtual usada en este proyecto. Disponible [Internet]: http://sourceforge.net/projects/virtualboximage/files/Ubuntu%20Linux/12.04/ubuntu_12.04-x86.7z/ [20 de Junio de 2013]

[64] Descarga conjunta del *JDK 7.11* más el *SDK* de *Java EE 6.4*. Oracle Corporation. Disponible [Internet]: http://www.oracle.com/technetwork/java/javaee/downloads/java-ee-sdk-6u4-jdk-7u11-downloads-1900532.html [20 de Junio de 2013]

[65] Descarga de MySQL Community Server. Disponible [Internet]: http://dev.mysql.com/downloads/mysql/ [20 de Junio de 2013]

[66] Tutorial de JavaScript. W3schools. Disponible [Internet]:
http://www.w3schools.com/js/js_intro.asp [20 de Junio de 2013]

[67] Formularios HTML. Ejemplos. Disponible [Internet]:
http://html.conclase.net/w3c/html401-es/interact/forms.html [20 de Junio de 2013]